パソコンプログラマのためのクリエイティブマガジン

平成3年4月2日第三種郵便物認可　1991年7月1日発行(毎月1回18日)第1巻第7号通巻7号

月刊プログラマーズ・ページ

Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

1991 7 JULY

定価980円



特集

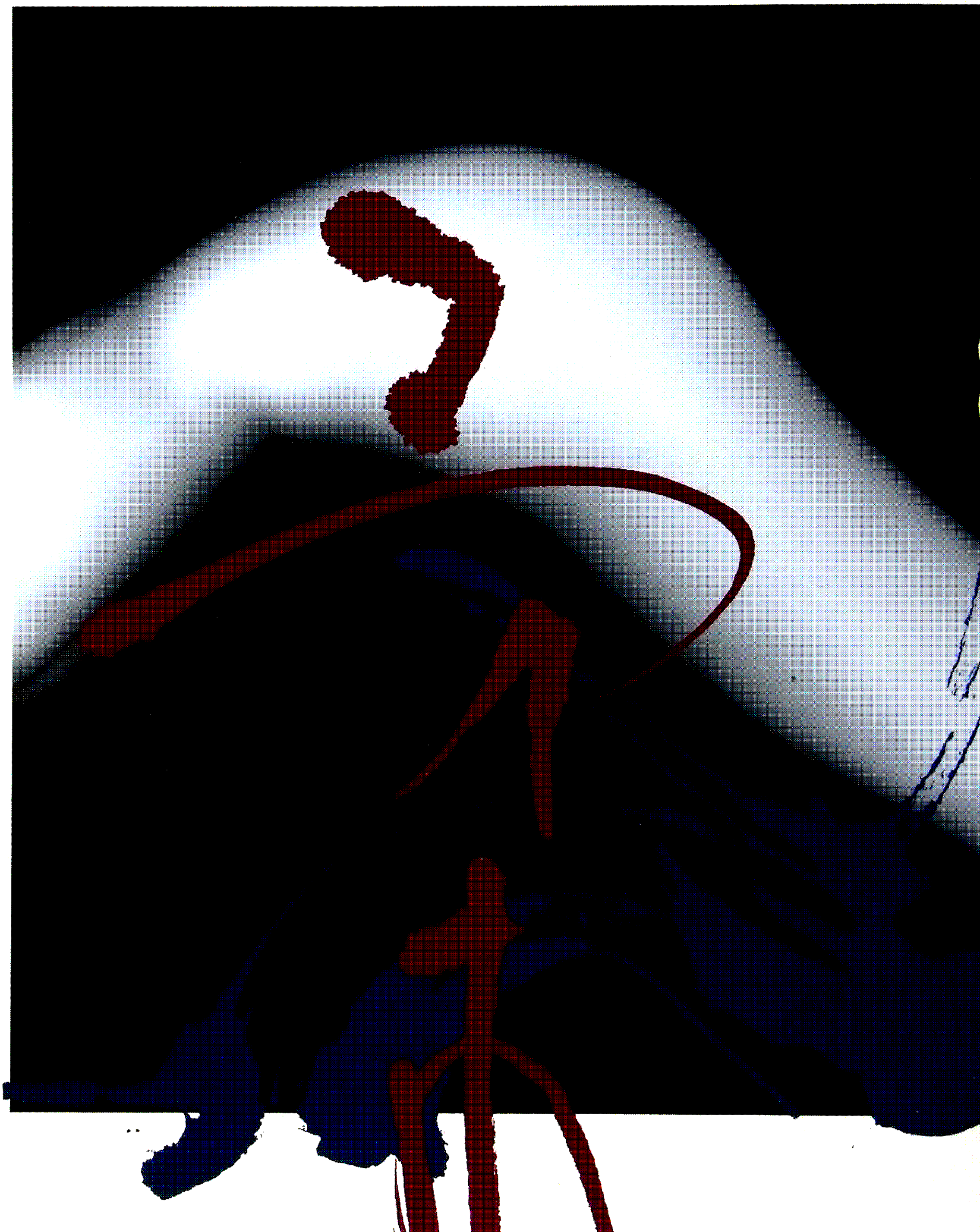
最新「デバイスドライバ」研究

MS-DOSデバイスドライバの技法と
新しいOS環境における基礎

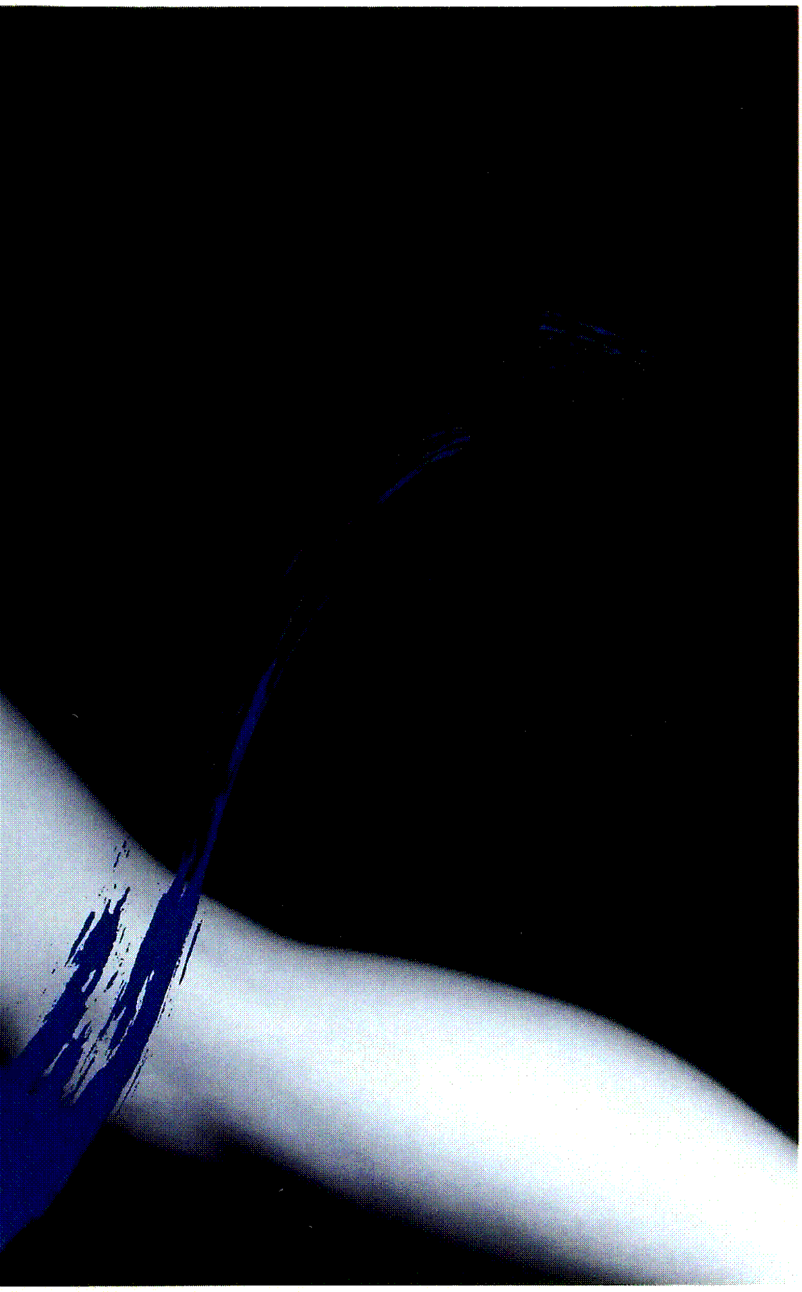
付録:5”2HDフロッピーディスク

本誌掲載プログラム／綴じ込み付録テキストファイル
（MS-DOS Version 3.3ファンクションリクエスト一覧）
／クイズ・ランダムスクリーン 解読ツール／通信ソフト
WTERM／ディスクコピーツールDCP

SE
Shoeisha Co., Ltd.



柔らかさを追求する
パーソナルコンピュータはAXです。



BORLAND
The Leader in Object-Oriented Programming

問：Windows™アプリケーション開発に最も有効なコンパイラは、何ですか？

（解答はページ下をご覧下さい）

(Inactive C:¥WORK¥BORLAND.EXE)

	従来の開発環境	?
Cコンパイラ	¥ 98,000	¥ 98,000
C++コンパイラ	なし	含む
Windows AP作成ツール	¥ 98,000	含む
アセンブラ	¥ 40,000	含む
合計	¥236,000	¥ 98,000

※上記の価格には、消費税は含まれていません。

パソコン上に、視覚的で快適な操作環境を提供するWindows 3.0。そこで利用するアプリケーション作成のためのコンパイラは、現在のものでベストといえるでしょうか？機能、スピード、価格…。これらの面で理想的ともいえるコンパイラが、登場しました。「BORLAND C++」です。BORLAND C++は、Windowsアプリケーションを開発、デバックするC/C++コンパイラとツールを標準で装備しています。

■Windowsアプリケーション開発をサポート

BORLAND C++は、Windowsプログラムを完全にサポート。高速かつ高性能な処理を実現しています。MDI(Multiple Document Interface)やDLLs(Dynamic Link Libraries)といったようなWindowsの機能を使って、大規模なプロテクトモードアプリケーションの開発が可能になります。さらに、MicrosoftのリソースコンパイラとWINDOWS.H、Resource Work Shop（英語版）*¹が標準装備されていますので、別途の開発キット(SDK)は不要です。

■ANSI C／AT&T C++両用コンパイラ

BORLAND C++は、CとC++両方のコンパイラとして利用できます。さらに、再コンパイルの速度を大幅に向上させる「コンパイル済みヘッダ」機能をサポート。MS-DOSの640KBというメモリの制限を破るオーバーレイマネージャVROOMの利用や、Turbo Driveによってプロテクトモードで動作する統合開発環境での大規模なプログラムの作成も可能です。

■Programmer's Platform

Programmer's Platformは、好きなツールを組み込む機能を持つ強力な統合開発環境で、MS-DOSやWindowsアプリケーション開発に必要なすべてのツールがすでに組み込まれています。さらにオーバーラップウインドウやマウス、そしてWindowsのプログラミングリファレンスを含んだオンラインヘルプをサポート。

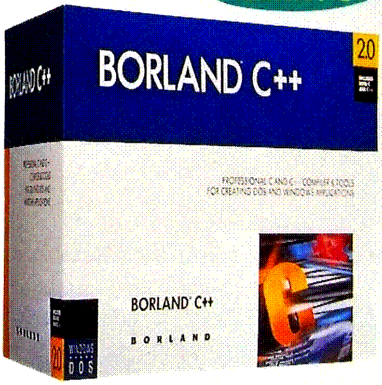
★お知らせ……BORLAND C++をお買い上げ・ユーザ登録された方には、9月登場予定の「Resource Work Shop」（ボーランド・オリジナルのWindowsアプリケーション開発用リソース編集ツール・日本語版）を無償にてお届けします。

■動作環境：PC-9801シリーズ（ハイレゾモード対応、LTは除く）／メモリ：640KB以上／MS-DOS：Ver 3.0以上／MS-Windows 3.0以上／マウス：バスマウス／要ハードディスク

Turbo C 2.0／Turbo C++1.0登録ユーザの方にはアップデートのご案内をお送りします。

価格……………………………98,000円（税別）

近日発売予定



（答） BORLAND® C++

*注…Resource Work Shop・英語版は、リソースとして日本語の文字列が扱えません。日本語版は、現在開発中です。
*All Borland Products are trademarks or registered trademarks of Borland International, Inc. ©Borland
*Microsoft、Windowsは米国マイクロソフト社の登録商標です。その他、商品名は一般に各社の商標です。
●製品に関するお問い合せは、総販売元㈱マイクロソフトウェアアソシエイツ TEL.03-3486-1411までお願いします。

株式会社ボーランドジャパン
〒107 東京都港区南青山7-8-1 小田急南青山ビル TEL.03-3486-1400代

〈資料請求番号001〉

Microsoft

Windows 3.0対応

日本語入力フロントプロセッサ　PC98シリーズ用

VJE-γ

Ver.1.0　ガンマ

■標準価格18,000円　絶賛発売中

ことばは操るものですぞ。



魔女はVJE-γの、水晶玉は辞書のアイコンです。秘宝は、VJE-γを相当使いこなしたときに現れます。

パソコンに向かっても、「私の日本語」で文章を綴っていけるようにしよう。
そのためには、たびたび使うことばは略語変換させる、自分専用のことばは辞書にのせるなどなどして、「私の辞書」をつくっておくとよい。
これらのことが、VJE-γならアイコンを使っていとも簡単にできてしまう。
もちろん、Windows 3.0にγを組み込むのも、操作環境を設定するのも、内容が手に取るようにわかるグラフィックな画面上から行える。
初めてパソコンを使う人も「私の日本語」を自由に操れるわけが、γにはある。

■ご注意■VJE-γ ver.1.0はNEC製のMicrosoft-Windows ver.3.0のプロセスとして動作するために、MS-DOS上では動作いたしません。DOSボックスおよびMS-DOS上では別途VJE-βが必要となります。

㈱バックス
〒194 東京都町田市森野1-34-10 第1矢沢ビル
インフォメーションセンタ ☎0427(24)9200

※詳しい内容については資料請求あり次第、お送りいたします。

※MS-DOS・Microsoft-Windowsは米国マイクロソフト社の登録商標です。

〈資料請求番号002〉

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

特集

最新「デバイスドライバ」研究

MS-DOSデバイスドライバの技法と新しいOS環境における基礎

表紙イラスト：日暮修一
表紙AD：佐々木幹夫(SE)

1991 JULY No.7

Brief

Full Screen Editor 日本語版

全米のプログラマが認めた実力！
アメリカでは「ブリーフ」が
プログラミング用エディターとして
最も多く利用されています。

C言語で組めるマクロ！

Version UP!

v.3.0 いよいよ登場。
プログラミングエディター新世紀。



よりパワフルで、よりエレガントな プログラミングのために……。

●C言語で組めるマクロ 新機能

使い慣れているC言語でマクロが組めますから、習得が簡単でカスタマイズも思いのままです。

●マクロのスクリーン・デバッガ 新機能

ソースコードでデバックできます。もちろんブレークポイントやウォッチ・ドッグもセットできます。

●テンプレート機能〈構文の自動生成〉 新機能

C、Ada、Basic、COBOL、FORTRAN、MODULA2、Pascalで構文を自動生成します。

●300回ものアンドゥ

各ファイルごとに300までの操作を元にもどすことができますので、やり直しの面倒を気にせず、安心して編集できます。

●安全編集

Briefは、一定時間キーを操作しないこと最新の編集内容をファイルに自動保存します。だから編集中に電源が切れても安心です。

●その他の主な新機能

キー操作のファイルへのセーブとリプレイ、処理可能行数を拡張、リジューム機能、Ctrlキー・アサイン(オプション)、メニュー(オプション)etc.

アメリカでも日本でも高い評価

●プログラミング向きエディターを探している人は、もう探す必要はない。(Jerry Pournelle/BYTE)
●ソフト開発には強力なツール、300回のアンドゥは大きな特徴。(日経バイト)
●既存のエディターに不満を覚えたら、これしかない。(Oh! FM)
●まるで夢のようなアンドゥだ——おもわず間違うのを期待してしまいそうだ。(The C Users Journal)

■主な機能

◆55ウィンドウ◆無限のファイル◆自動コンパイル◆自動エラー・ジャンプ◆自動インデント◆自動テンプレート◆バッファ書出 切換 削除 リスト表示◆ファイル名の指定 表示 変更◆バックアップ保存◆特殊文字入力◆2行を1行に連結◆ファイル単位の挿入◆増分検索◆各種マーク 始・終点の入換◆ブロックのカット/コピー/ペースト 印刷 保存 左右へのシフト◆対応する括弧へ移動◆C言語のコメント化/非コメント化◆DOSコマンドの実行◆キー操作記憶 実行◆入力ヒストリー◆英文テキスト整形◆タブのスペース挿入◆拡張子毎に異なった設定ができるタブ◆検索置換パターンの画面からの入力◆英字の大小文字変換 その他

定価 48,000円
(消費税は含まず)

■5機種対応パッケージ

(東芝、NEC、AX協議会、IBM、富士通)

●5"2DD、3.5"2DDデュアルメディア●MS-DOS3.1以降に対応●英語DOSでは使用できませんが、英語版BRIEFもセット販売いたしております。●お急ぎの方、上記以外のメディア希望の方、英語BRIEFのセット販売希望の方はユーザー・サポート係までご連絡下さい。

日本システム株式会社

●お問合せは 東京都立川市曙町1-18-2一清ビル立川事業所Briefユーザー・サポート係 TEL0425-23-4469 FAX0425-27-6127



●特集●

最新「デバイスドライバ」研究

MS-DOSデバイスドライバの技法と新しいOS環境における基礎

パーソナルコンピュータでは，外部ディスクドライブやプリンタ，モデムなどを自由に接続し，利用することが可能になっている．このようなシステムの拡張性を実現しているのが，デバイスドライバと呼ばれるプログラムである．このデバイスドライバが，ハードウェアとオペレーティングシステムの間でデータの入出力を制御することで，オペレーティングシステムのデバイス独立を実現している．今回の特集では，MS-DOSにおけるデバイスドライバの構造とプログラミングテクニックを中心に，新しいシステムとして注目を集めているMS-Windows 3.0，さらに次世代の標準OSといわれるOS/2におけるデバイスドライバついて，MS-DOSとの違いを明らかにするとともにその概念と構造について解説を行った．

DEVICE DRIVER

特集
最新「デバイスドライバ」研究

デバイスドライバ入門

入出力装置とのインターフェースを確立する

日笠　健　株式会社エー・ピー・ラボ



日頃，MS-DOS パーソナルコンピュータを扱うユーザーの多くは日本語 FEP，RAMDISK など，本来の MS-DOS には組み込まれていない基本機能の恩恵を受けている．これらの機能は周知の通り，「デバイスドライバ」という形式のプログラムによって実現されている．多くのユーザーにとって，デバイスドライバとはこれらの機能の組み込み時にのみお目にかかる言葉であろう．しかしながら，実際にデバイスドライバのプログラミングを行おうとするとき，その情報量の少なさには驚かされるものがある．また，デバイスドライバは何を目的とし，どのように動作すべきものなのか，について触れている文献も少ない．本稿ではまずデバイスドライバの概念を紹介し，MS-DOS デバイスドライバプログラミングを試みる際の基礎知識を養うことを目的としている．

はじめに

デバイスドライバの位置づけ

パーソナルコンピュータに限らず，コンピュータにはなんらかの入出力装置が必要である．たとえば，数値や文字を入力するためのキーボード，コンピュータの処理結果を表示するためのディスプレイ（ここでは，キーボードとディスプレイを合わせて「コンソール」と呼ぶ），印刷用のプリンタ，フロッピーディスクやハードディスクなどの外部記憶装置，GUIには欠かせないマウス，シリアル通信回線，そのほか……．日頃使っているパーソナルコンピュータでさえ，5本の指に余るほどの入出力装置を標準で装備している．これらの標準的な入出力装置以外にも，イメージスキャナ，音声合成装置，磁気テープなど用途と予算に応じて自由に増設することができる．また不要になれば取り外すことも可能である．

しかしながら，入出力装置というハードウェアだけを用意したところで，コンピュータ内部のCPUから利用できないのでは意味がない．さまざまな外部のハードウェア装置とCPUの仲立ちをとる仕組みが必要である．この仕組みが「インターフェース」であり，ハードウェアおよびソフトウェア両面にわたって接続のためのルールが決められている．たとえば，RS-232Cという規格に準拠したインターフェースを備えているコンピュータには，市販されているほとんどのモデム装置を接続することができる．そして，いくつかの通信用パラメータさえ正しく設定すれば，通信用のアプリケーションプログラムによってモデムを介した通信が可能である．

ただ，「インターフェース」という言葉は接続用のコネクタ形状や，電気的条件だけを意味するものではない．アプリケーションプログラムの立場から見れば，どのような方法で通信回線に1文字送信できるのか，あるいは，通信回線から1文字取得できるのか，が重要となる．数種類のハードウェア上で動作するプログラムを記述しようとする場合，アプリケーションプログラムから見た入出力の方法は，機種に依存しない一定の「プログラムインターフェース」にしたがっている必要がある．たとえ単独機種用のプログラムであっても，汎用的なプログラムインターフェースを確立しておけば，二度目以降のプログラミングはかなり楽になる．

RS-232Cを例にとれば，1文字入出力のほかにも，ボーレイト，文字長，パリティといった通信パラメータの定義，回線ステータスの取得などの方法が統一されていることによるメリットは大きい．

プログラムインターフェースを確立するうえで，もっとも入出力装置（デバイス）に近い部分の処理を担当するソフトウェアが（狭義の）デバイスドライバである．デバイスドライバは，より上位のソフトウェアからの要求に応じて，インターフェース・ハードウェアを制御するための入出力命令を実行し，データの書き込みや読み出しを実行する．

MS-DOS システムコールとキャラクタデバイス

MS-DOSが扱うデバイスは，大きく分けてキャラクタデバイスとブロックデバイスの2種類である．キャラクタデバイスは，キーボードのような文字単位での入出力を行い，ブロックデバイスは，ディスク装置のようにブロック単位で入出力を行う．ここでは，アプリケーションプログラムの立場から見たデバイスの位置づけを理解するために，MS-DOSのシステムコールとキャラクタデバイス用デバイスドライバの関係を考えてみる．

●●● MS-DOS システムコールの概略

デバイスドライバの位置づけを理解しようとする際に，デバイスドライバに作用するシステムコールを知ることは重要である．特殊なデバイスを扱うものでないかぎり，アプリケーションプログラムからデバイスドライバを直接呼び出すことは稀であり，ほとんどの場合，何らかのシステムコールによってデバイスドライバに動作を促すことになる．

MS-DOSのシステムコールをおおまかに分類すると以下のようになる．

- 標準キャラクタデバイスI/O
- メモリおよびプロセス管理
- ファイル・ディレクトリ管理
- そのほか

これらのシステムコールは，ほとんどが“INT 21H”というソフトウェア割り込みによって利用可能であり，キーボードやディスプレイなどのハードウェアを操作する機能であっても，動

作しているハードウェア機種に依存しない．これらのシステムコールとハードウェアの関係を考えてみる．

標準キャラクタデバイスI/O

初期のMS-DOSから用意されているシステムコール群であり，基本的には1文字単位にデバイスとの入出力を行う．種々のモードが用意されたコンソール（CON）入出力（キーボード入力とディスプレイ出力），補助(AUX)入出力(多くの場合，RS-232Cインタ－フェースとの入出力)，プリンタ（PRN）出力が用意されている．

これらのシステムコールは，CP/Mに用意されていたBDOSコールとほぼ互換性がある(CP/Mについてはコラム参照)．

メモリおよびプロセス管理

MS-DOSには，複数のプロセスをメモリに配置したり，あるプロセスが利用するメモリ領域を割り当てるための機能が用意されている．割り当てられたメモリ領域は，メモリコントロールブロックによって管理される．

ファイル・ディレクトリ管理

MS-DOS Ver.2.0以降はUNIXの影響下にある，といわれている．その理由は，階層ディレクトリ構造とファイルハンドルを用いたストリーム入出力にある．ディレクトリ構造については本稿では触れていない．

MS-DOSは，データの入出力を単純なバイト列（ストリーム）として扱う．つまりレコードや構造体単位でのデータ入出力用のシステムコールは用意されておらず，すべての入出力は，データを格納するバッファのポインタと入出力バイト数で規定される．旧バージョンとの互換性のために，FCBやDTAを用いた入出力用システムコールも用意されてはいるが，互換性の維持以上の意味はない．

ファイルとは，MS-DOSの内部とアプリケーションプログラムをつなぐパイプのようなものである．そのパイプの中をデータがバイト単位に流れていく．ファイルハンドルは，パイプを分類するための名札のようなものである．アプリケーションプログラムから見れば，あるファイル名で書き込んだ（ディスク）ファイルがどのように格納されるのか，を知る必要はない．また，ある名前でオープンし読み出しているファイルが，実際にどこに格納されているのか，を知っている必要もないのである．

標準ハンドルとデバイス

標準キャラクタデバイスI/O用システムコールには，CP/Mと同様に3種類のキャラクタデバイスとの入出力が用意されているが，Ver.2.0以降のMS-DOSでは，これらのキャラクタ

COLUMN

CP/Mのモジュール構成

CP/Mは，1970年代の後半から使われている，8080，Z80などのインテル系8ビットCPU用の標準的なDOSである．1980年代のはじめに登場した初期のMS-DOSは，CP/Mとの互換性をある程度は売りものにしていた．

MS-DOSが86系CPU専用のDOSであるのに対し，CP/Mは8080用からはじまって，8086系用，68000用など各種のCPU向けに移植された．

CP/Mは，以下のようなソフトウェア・モジュールから構成されている．

CCP(Console Command Processor)

・MS-DOSでのCOMMAND.COMに相当する．
・入力されたコマンドの解析と実行．

コマンドにはCCP内に組み込まれているビルトインコマンドと，実行要求にのたびにディスクからメモリ（TPA：Transient Program Area）に読み込まれるトランジェントコマンドの2種類がある．

BDOS(Basic Disk Operating System)

・MS-DOSでのMSDOS.SYSに相当する．
・アプリケーションプログラムからのファンクションコール(BDOSコール)に応じた処理の実行およびBIOSへの分岐処理を行う．さらに，ディスク領域とファイルの管理，キャラクタI/O処理などを実行する．

BIOS(Basic I/O System)

・MS-DOSでのIO.SYSおよび各デバイスドライバに相当する．
・動作するハードウェアにあわせたハードウェア制御ルーチン群．

MS-DOSでは，各機能を担当するモジュールは独立したファイルとして扱われるが，CP/Mではシステム生成用のコマンドによってメモリへのロードイメージがディスクに直接書き込まれる．したがって，コンソール入出力装置を変更する場合など，BIOS部分のプログラムを変更し，アセンブル後に再インストールする必要がある．

NEC PC-98シリーズ一覧
PC-9801（1982/10発表）●CPU:8086(5MHz)●メインメモリ：128KB●スロット数：5●備考：アナログ16色不可，FDDなし，漢字ROM・マウスI/Fオプション

表 1　標準キャラクタデバイスのファイルハンドル

番号	ハンドル名	デバイス
0	標準入力(stdin)	CON (入力)
1	標準出力(stdout)	CON (出力)
2	エラー出力(stderr)	CON (出力)
3	補助(stdaux)	AUX (入出力)
4	プリンタ(stdprn)	PRN (出力)

入出力もまたファイルハンドルに基づいて行われる．つまり，(ディスク)ファイルとキャラクタデバイスは，入出力時には同等に扱われる．あるデバイスをオープンすることによって，デバイスの入出力をファイルハンドルに基づいて行うことができるのである．

標準キャラクタデバイスについては，表1の5つのファイルハンドル（標準ハンドル）があらかじめオープンされ，割り当てられている．

COMMAND.COMのプロンプトに対して，

A>dir > file

というコマンドを入力した場合，“DIR”コマンドの実行結果が“FILE”という名前の（ディスク）ファイルに書き込まれるのは周知の通りである．この処理は「リダイレクト」として知られている．リダイレクトを行わない場合には，“DIR”コマンドの実行結果は標準出力（多くの場合ディスプレイ）に表示されることになる．リダイレクト処理とは，ストリームの出口を“>”記号の後ろにある（ディスク）ファイルに切り替えることである．MS-DOSの標準入出力は，リダイレクトが可能である．

●●● デバイスによる入出力の一般化

MS-DOSにあらかじめ用意されている標準デバイスを用いることによって，基本的な入出力に必要なコンソール，プリンタ，シリアル回線に簡単にアクセスすることができる．ハードウェアインターフェースについての知識はほとんど不要であり（シリアル回線については一考を要するが），同時にプログラミングにおいても通常のMS-DOSシステムコールですべてをまかなうことができる．

アプリケーションプログラムの立場から見れば，キャラクタデバイスの入出力においてデバイスドライバを意識する必要はない．プログラムから「見える」のは（ディスク）ファイルのファイル名に相当するデバイス名のみなのである．ブロックデバイスにいたっては，アプリケーションプログラムからは何も見えない．

MS-DOS デバイスドライバの実際

前章では，アプリケーションプログラムから見たデバイスドライバについて述べたが，ここではデバイスドライバの構造，MS-DOS(MSDOS.SYS)とデバイスドライバの関係についてキャラクタデバイスを中心に解説する．

※以降の章に示すデバイスヘッダ情報やデバイスリクエストファンクションは，NECのPC-9800シリーズ用MS-DOS V.3.3Cに付属のマニュアルを基準としている．旧バージョン（とくにVer.2.0）ではサポートされていない機能もあるため，ご使用のMS-DOSに添付のマニュアルで確認していただきたい．

● 組み込み済みデバイスとインストール可能デバイス

デバイスドライバには，キャラクタデバイスとブロックデバイスの2種類があると同時に，MS-DOSシステム内にデバイスドライバを組み込む方法から見ても2種類に分けることができる．それは，IO.SYS内にあらかじめ用意されている「組み込み済み」デバイスドライバと，CONFIG.SYS内（あるいはADDDRV.EXEで利用するデバイス定義ファイル内）の“DEVICE =”文によってシステムへの組み込みを行う「インストール可能な」デバイスドライバの2種類である．

これら2つのデバイスドライバの違いは，表2のとおりである．

組み込み済みデバイスとインストール可能なデバイスの間に機能的な差はない．組み込み済みデバイスの目的は，

表 2　組み込み済みデバイスとインストール可能なデバイスの違い

	組み込み済みデバイス	インストール可能なデバイス
構成ファイル	IO.SYS	任意（※）
システムからの切り離し	不可能	可能
柔軟性	低い	高い

※インストール可能なデバイスは，通常の場合1つのファイルに1つのデバイスドライバが記述されている．しかしながら，1つのファイル内に複数のデバイスを記述することも可能である．本文後半に例を示している．

MS-DOSの基本的な動作に必要な5つの標準デバイス（CON，PRN，AUX，CLOCK，NUL）と，接続されているディスクを扱うためのブロックデバイスを提供することにある．これらのデバイスが存在しない場合，MS-DOSは先に述べた標準キャラクタ入出力やディスク入出力を実行することができない．

デバイスドライバの構造

MS-DOS用デバイスドライバを作成する際，特別な開発環境は不要である．先頭に「デバイスヘッダ」と呼ばれる構造をもつバイナリファイルであれば，COM形式，EXE形式のいずれでもかまわない．デバイスヘッダは，そのデバイスの名前や属性，MS-DOSがデバイスを管理するために必要な情報，MS-DOSからデバイスへの実行要求（「デバイスリクエスト」と称する）に必要な情報などを保持しており，非常に重要な役割を果している．

デバイスヘッダを，MS-DOSとデバイスドライバのインターフェース情報と考えることもできる．MS-DOSは，デバイスヘッダ内の情報に基づいて任意のデバイスの名前や機能を知り，適切なデバイスリクエストを発行することで，アプリケーションプログラムからのシステムコールを処理する．MS-DOSとデバイス間の正しいインターフェース処理を行うためのいくつかの規約さえ押さえておけば，デバイスドライバのプログラミングは容易である．

図1 デバイスドライバの構造

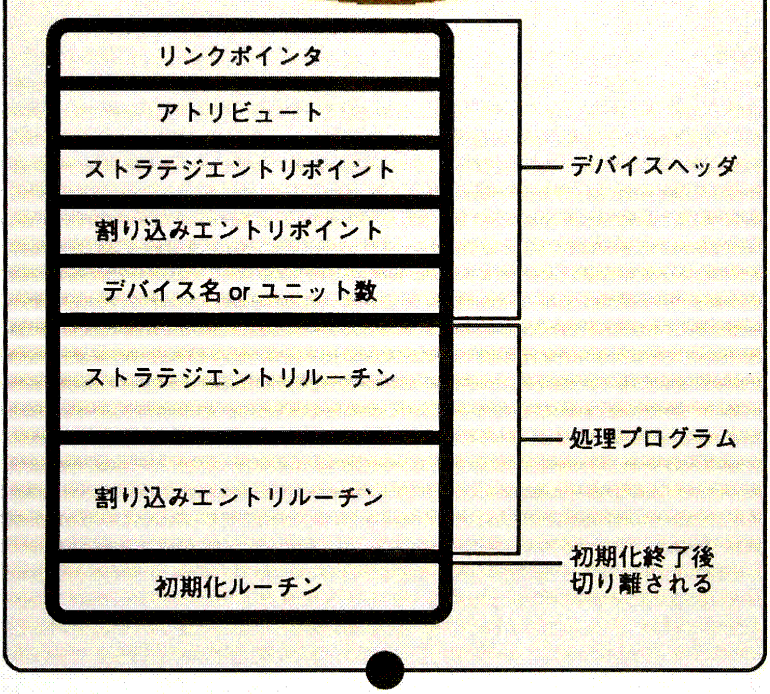


図2 デバイスヘッダの構造

オフセット	名称（サイズ）	目的と機能
0	デバイスリンク（dword）	次のデバイスへのポインタ．作成時には－1としておく．
4	デバイスアトリビュート（word）	このデバイスの属性を示す．
6	デバイスストラテジポインタ（word）	デバイスストラテジルーチンのエントリアドレス（オフセットアドレス）
8	デバイス割り込みポインタ（word）	デバイス割り込みルーチンのエントリアドレス（オフセットアドレス）
A	デバイス名（8バイト）	キャラクタデバイスの場合にはデバイス名 ブロックデバイスの場合にはユニット数．

デバイスヘッダ

デバイスヘッダの構造を図2に示す．

●●● デバイスリンク

このフィールドは，MS-DOSがメモリ内にある複数のデバイスを管理するために用いられる．また，複数のデバイスヘッダを備えたプログラムでは，このフィールドにあらかじめ初期値をセットしておくことにより，複数デバイスを組み込む必要がある旨をMS-DOSに通知することができる．

あるデバイスがMS-DOSに組み込まれると，そのデバイスのデバイスリンクフィールドには，直前に組み込まれたデバイスのデバイスヘッダアドレスが書き込まれる．このようにしていくつかのデバイスが組み込まれた後，最後に組み込まれたデバイスから順に

デバイスリンク内のアドレスをたどっていけば，すべてのデバイスヘッダの位置を知ることができる．つまり，デバイスヘッダのリスト（デバイスリンクチェーン）が作成されるのである．

このリストは，－1という値をもつデバイスヘッダで終了し，NULデバイスのデバイスリンク内に，リストの先頭にあるデバイスヘッダ（最後に組み込まれたデバイスヘッダ）のアドレスが保持されている．

デバイスドライバを作成する際には，このフィールドに初期値として－1（FFFF:FFFF）をセットする．ただし，1つのプログラム内に複数のデバイスヘッダを用意する場合には，少々気を使う必要がある．これについては，「複数のデバイスヘッダをもつデバイスドライバ」項でサンプルプログラムを用いて解説している．

●●● デバイスアトリビュート

デバイスが扱うデータのタイプや，デバイスドライバの内部動作に関する情報を設定するためのフィールドであり，16ビットのフラグから構成されている．表3にデバイスアトリビュートのビット構成を示す．

デバイスアトリビュートを構成する各ビットの詳細については，参考文献を参照していただきたい．ここでは，いくつかの重要なビットについて解説する．

●●● デバイス属性ビット（ビット15）

このビットは，デバイスドライバが扱うデータの種別を示す．キャラクタデバイスは，バイト単位の入出力を制御し，ブロックデバイスはある程度の大きさ（ディスクのセクタサイズなど）にブロック化されたデータを扱う．

表3　デバイスアトリビュートのビット構成

ビット		意　味
15（MSB）		デバイス属性ビット（1：キャラクタ，0：ブロック）
14		IOCTLビット（1：IOCTLサポート，0：非サポート）
13	キャラクタ	OUTPUT UNTIL BUSYビット（1：サポート，0：非サポート）
	ブロック	NON FAT IDビット
12		（予約域）　0をセットする．
11		OPEN/CLOSE/REMOVABLEビット（1：サポート，0：非サポート）
10〜7		（予約域）　0をセットする．
6		V3.3ビット（1：V3.3対応，0：非対応）
5		（予約域）　0をセットする．
4〜0	ブロック	（予約域）　0をセットする．
4	キャラクタ	特殊CONデバイスビット
3	キャラクタ	カレントCLOCKデバイスビット
2	キャラクタ	カレントNULデバイスビット
1	キャラクタ	標準出力デバイスビット
0	キャラクタ	標準入力デバイスビット

キャラクタデバイスの場合，デバイスヘッダ内に「デバイス名」をセットする．アプリケーションプログラムは，デバイス名を指定しファイルとしてオープンすることにより入出力を行うキャラクタデバイスを指定することができる．

ブロックデバイスには名前がないため，キャラクタデバイスのように入出力の対象として特定のデバイスドライバをアプリケーションプログラムから指定することはできない．組み込まれたあとのブロックデバイスは，MS-DOSからドライブ名（A：，B：……）で識別される．ファイルのオープン時にアプリケーションプログラムから渡されたディレクトリ名やファイル名は，MS-DOSの内部で対応するドライブに結びつけられる．そして，アプリケーションプログラムが入出力のシステムコールを実行した際に，アプリケーション側のストリーム入出力がブロックデバイスのサポートするセクタ単位の入出力に変換され，デバイスリクエストが実行される．

●●● IOCTLビット（ビット14）

IOCTL（I/Oコントロール）とは，組み込まれているデバイスの動作モードを変更したり，デバイスが制御するハードウェアの動作状態を調べるなどの目的で行う，直接的なデバイス制御である．基本的には本来のデータの入出力に用いるべきではない．

たとえば，RS-232Cインターフェースを制御するAUXデバイスに対して，ボーレイト，パリティビットなどの通信パラメータの設定および参照を行いたいとする．IOCTL機能が用意されていない場合，アプリケーションからMS-DOS経由で（システムコールによって）デバイスにアクセスする手段は，AUXデバイスを介したデータの入出力以外にはない．このような場合，通信パラメータなどのようなハードウェアに依存する項目の設定や参照は，いわゆるROM BIOSを用いるか，あるいは，ハードウェアに直接アクセスする以外にない．

IOCTLをサポートするデバイスは，MS-DOSからのIOCTLリクエストを処理するルーチンを用意しておく必要がある．このルーチン内ではMS-DOS経由でアプリケーションプログラムとの制御情報やデータの交換を行い，それにしたがってデバイスの動作を変更することができる．

特殊 CON デバイスビット（ビット 4）

このビットは，標準出力デバイスビット（ビット 1）とともに用いる．このビットがセットされているデバイスに MS-DOS が文字を出力する場合，通常のデバイスリクエストの代わりにソフトウェア割り込み（"INT 29H"）が実行される（AL レジスタ内に出力する文字が入っている）．このため，このビットをセットするデバイスは，INT 29H 用の割り込みハンドラを用意する必要がある．

ただ，数年前に調べたところ，標準出力が RAW モードに設定されている場合の標準出力への文字出力は，通常のデバイスリクエストによって行われていた．このビットを利用する際には，十分な注意が必要である．

標準入出力デバイスビット

- カレント NUL デバイスビット（ビット 3）
- カレント CLOCK デバイスビット（ビット 2）
- 標準出力デバイスビット（ビット 1）
- 標準入力デバイスビット（ビット 0）

デバイス名に対応する MS-DOS の標準キャラクタデバイスを置き換える際に，これらのビットをセットする．ただし，新たな NUL デバイスを組み込んでも意味がない．

MS-DOS がアプリケーションプログラムから指定されたデバイス名をもつデバイスヘッダの探索を行う際にはデフォルトの NUL デバイスを振り出しに，一致するデバイス名が見つかるか，リンクチェーンの末尾（デバイスリンクが－1）に行き着くまでデバイスリンクチェーンたどっていく．したがって，デバイス名の一致するデバイスが見つかった時点で探索が終わるため，つねにリンクチェーンの先頭にあるデフォルトの NUL デバイスが置き換えられることはない．NUL 以外のデバイスの場合には，あとから組み込まれたデバイスが優先することになる．

ストラテジエントリポインタとデバイス割り込みエントリポインタ

MS-DOS でのデバイスリクエストとは，MS-DOS によるデバイスストラテジルーチンとデバイス割り込みルーチンの実行である．デバイスドライバの作成時には，デバイスヘッダ内のこれら 2 つのポインタに，それぞれのルーチンの 2 バイトのエントリアドレス（デバイスドライバの先頭からのオフセット）を記述しておく．MS-DOS はデバイスヘッダ内のこれら 2 つのポインタと，デバイスが組み込まれているセグメント値を基に 2 つのルーチンのアドレスを取得し，デバイスリクエストを実行する．

デバイス名

キャラクタデバイスではデバイス名，ブロックデバイスではユニット数をセットしておく．キャラクタデバイスは 8 バイトの文字列によってデバイス名を記述する．たとえば CON デバイスの場合，"CON　　　"（5 バイトの空白）のようにデバイス名のあとの領域についても空白を補っておく必要がある．

典型的なデバイスヘッダ

以上のようなデバイスヘッダの構造をアセンブリ言語で記述すると，リスト 1 のようになる．このデバイスヘッダは，デバイス名 "TEST0000" をもつキャラクタデバイスを表している．

リスト 1　典型的なデバイスヘッダ例

```
（プログラムの先頭であること）
header          dd      -1              ; デバイスリンクの初期値
                dw      08000h          ; デバイスアトリビュート
                dw      strategy        ; ストラテジエントリポインタ
                dw      intentry        ; 割込みエントリポインタ
                db      'TEST0000'      ; デバイス名

strategy        proc    far             ; デバイスストラテジルーチン
                ...
                ret                     ; セグメント間リターン
strategy        endp

intentry        proc    far             ; デバイス割込みルーチン
                ...
                ret                     ; セグメント間リターン
intentry        endp

                end
```

リスト 2　ストラテジルーチンの記述例

```
vector          struc
        ofs             dw      ?
        seg             dw      ?
vector          ends

ptrsave         dd      -1      リクエストヘッダ・アドレスの保存領域

strategy        proc    far     リクエストヘッダ・アドレスの保存のみを行う．
                mov     word ptr cs:ptrsave._ofs, bx
                mov     word ptr cs:ptrsave._seg, es
                ret
strategy        endp
```

デバイスリクエスト

MS-DOSは，アプリケーションプログラムからデバイス入出力を要求されると，アプリケーションから渡された情報（たとえば，オープンするデバイス名，転送アドレス，転送バイト数）を基に，対応するデバイスに対してデバイスリクエストを行う．デバイスリクエストを行う際，アプリケーションから渡された情報は，MS-DOS内でデバイスとインターフェースするためのデータ（「リクエストヘッダ」を先頭とする配列）に変換され，デバイスとやりとりされる．デバイスは，MS-DOSから渡されたリクエストヘッダへのポインタ（4バイト）を基に入出力のための情報を取得し，入出力の実行結果をこのポインタが指す構造体内に格納する．

デバイスリクエストは，具体的にはデバイスストラテジルーチンの実行と，それにひき続くデバイス割り込みルーチンの実行に分かれている．

デバイスストラテジルーチンの呼び出し

MS-DOSがデバイスリクエストを行う際には，対応するデバイスのストラテジルーチンをまず実行する．このルーチンでは，リクエストヘッダのポインタがMS-DOSからデバイス側にひき渡されるため，デバイス側ではこのポインタをデバイス内に保存しておく．

ストラテジルーチンおよびデバイス割り込みルーチンは，呼び出された時点とリターンする時点で全レジスタの内容が等しいことを保証しなければならない．また，各ルーチンの呼び出し時に使用可能なスタックの大きさは最少でも40バイトである．より多くのスタックを必要とする場合は，デバイス内でローカルスタックを用意する．

リクエストヘッダ

リクエストヘッダはMS-DOSが用意する13バイトの構造体であり，デバイスリクエストに際してデバイスとのやりとりに用いられるメモリ領域の先頭に位置する構造体である．リクエストヘッダは表4のような構成となって

表 4　リクエストヘッダの構成

オフセット	サイズ(バイト)	おもな目的
0	1	デバイスリクエスト用データ領域の大きさ（バイト単位）．
1	1	ブロックデバイス専用であり，処理を要求するユニット番号を指定．（そのブロックデバイス内でのローカルな番号）．
2	1	デバイスリクエストの処理を表すコマンドコード．
3	2	デバイス側がリクエスト処理結果のステータスを返す領域．
5	8	（予約域）

表 5　デバイスリクエストの一覧

リクエスト名	コマンドコード	条件
INIT	0	
MEDIA CHECK	1	ブロックデバイスのみ
BUILD BPB	2	ブロックデバイスのみ
IOCTL INPUT	3	IOCTLビット=1
INPUT	4	
非破壊INPUT	5	キャラクタデバイスのみ
INPUT STATUS	6	キャラクタデバイスのみ
INPUT FLASH	7	キャラクタデバイスのみ
OUTPUT	8	
VERIFY付きOUTPUT	9	
OUTPUT STATUS	10	キャラクタデバイスのみ
OUTPUT FLASH	11	キャラクタデバイスのみ
IOCTL OUTPUT	12	IOCTLビット=1
DEVICE OPEN	13	OPEN/CLOSE/REMOVABLEビット=1
DEVICE CLOSE	14	OPEN/CLOSE/REMOVABLEビット=1
REMOVABLE MEDIA	15	OPEN/CLOSE/REMOVABLEビット=1，ブロックデバイス
OUTPUT UNTIL BUSY	16	OU PUT UNTIL BUSYビット=1，キャラクタデバイス
GENERIC IOCTL	19	V3.3ビット=1
DEINSTALL	20	キャラクタデバイス
GET DRIVE MAP	23	V3.3ビット=1
SET DRIVE MAP	24	V3.3ビット=1

リスト 3　サンプルプログラムで利用している構造体

```
reqheader       struc
        _length         db      ?
        _unit           db      ?
        _cmd            db      ?
        _status         dw      ?
        _reserve        db      8 dup (?)
        _media          db      ?
        _address        dd      ?
        _count          dw      ?
        _sector         dw      ?
        _volid          dd      ?
reqheader       ends

reqheader2      struc
                        db      13 dup (?)
        _unitnum        db      ?
                        dd      ?
        _bpbptr         dd      ?
        _devnum         db      ?
reqheader2      ends
```

いる．

リクエストヘッダ以降のメモリに格納される情報は，デバイスリクエストの種類に応じて変化するが，キャラクタデバイス用とブロックデバイス用の2種類に大別できる．本稿のサンプルプログラムでは，リスト3のような構造体を定義し利用している．ここに定義されている構造体のメンバ名によってリクエストヘッダへのポインタを修飾し，アクセスすべきアドレスとしている．

表 6　終了ステータス

ビット	名称	意味
15（MSB）	ERROR	リクエスト処理中にエラーが発生．（ビット7〜0にエラーコードを格納する）
14〜10	（予約域）	（未使用）
9	BUSY	非破壊INPUTおよびINPUT STATUSリクエスト時に，入力すべき文字がないことを知らせる．
	REMOVABLE MEDIA	リクエストの実行結果を返す．
8	DON	デバイスリクエストの処理が終了した．
7〜0	ERROR=1のとき，	以下のようなエラーコードを格納する．

エラーコード	エラーの内容
00H	ライトプロテクトされているメディアへの書込み
01H	存在しないユニット番号の指定
02H	ドライブの準備ができていない
03H	無効なコマンド
04H	CRCエラー
05H	ドライブリクエストの長さがまちがっている
06H	シークエラーの発生
07H	無効なメディアの指定
08H	指定のセクタが存在しない
09H	プリンタの用紙切れ
0AH	書き込みエラー
0BH	読み込みエラー
0CH	一般的なエラーの発生
0DH	（未使用）
0EH	（未使用）
0FH	不正なディスク交換

リスト 4　デバイス割り込みルーチンの記述例

```
intentry        proc    far
                pushf
                irp     reg,<ax,bx,cx,dx,si,di,ds,es>
                  push  reg                         使用する全レジスタを保存する.
                endm    .
                mov     cs:error_code, 0            エラーコードの初期値
                les     bx, cs:ptrsave              リクエストヘッダへのポインタ
                mov     al, es:[bx]._cmd            コマンドコードを得る.
                ...
コマンドコードの解析と，各リクエスト処理ルーチンへの分岐を行う．
DONEビット以外のステータスは，各処理ルーチン内でAHレジスタにセットする．
                ...
exit:           or      ah, 1                       DONEビットをセット
                mov     al, cs:error_code           エラーコード
                mov     es:[bx]._status, ax         終了ステータスの格納
                irp     reg,<es,ds,di,si,dx,cx,bx,ax>
                  pop   reg                         全レジスタの復帰
                endm
                popf
                ret
error_code      db      0
intentry        endp
```

たとえば，ES:BXにリクエストヘッダへのポインタが入っている場合，コマンドコードをALレジスタに読み込むためには，

```
mov  al, es:[bx]._cmd
```

とする．また，「ストラテジルーチンの記述例」内にある“vector”という構造体もサンプル内でひんぱんに用いている．筆者は，アドレス変数を構成するセグメントとオフセットのメモリ内での順序をしばしばわすれてしまう．情けないまちがいをしないための安全上の習慣として用いている．この構造体の使用例は以下のようになる．

```
mov  ax, es:[bx]._address_ofs
```

この記述によって，AXレジスタ内にアドレスフィールド中のオフセット部を得ることができる．

●●● デバイス割り込みルーチンの呼び出し

デバイスリクエストに対する具体的な処理は，デバイス割り込みルーチンで行われる．このルーチンでは，先にストラテジルーチンが保存したリクエストヘッダへのポインタ（ptrsaveの内容）を基にコマンドコード（_cmd）を参照し，処理すべきリクエストの種類を知る．そしてそれに対応した処理を行った後，必要に応じて処理結果をリクエストヘッダにひき続くデータ領域（_addressが示す領域）に転送し，終了ステータス（_status）をセットする．また，処理中にエラーが起きた場合にはステータスフィールド内にエラーコードをセットする．

表5にデバイスリクエストとそれに対応するコマンドコードの一覧を示す．

NEC PC-98シリーズ一覧
PC-9801U2（1985/5発表）●CPU:V30（8MHz）●メインメモリ：128KB●スロット数：2●備考：アナログ16色オプション，GVRAM1ページ

終了ステータスは表6のような構成の1ワードのデータである.

デバイス割り込みルーチンは，おおまかにはリスト4のような処理となる.

各コマンドコードの詳細については参考文献を参照していただきたい．ここでは，サンプログラム内で扱っているINITリクエストについて解説する.

キャラクタデバイスでのINITリクエスト

INITリクエストは，デバイスドライバの組み込み時に一度だけ発行される．このリクエストの目的は,

- デバイスが動作する環境(DOSのバージョン，残りメモリ量，機種)の確認
- デバイスが変数領域などに利用するメモリの初期化
- ディスクI/Fなど物理的な入出力を行うためのハードウェアの初期設定

などである．また，通常の場合は一度メモリに常駐したデバイスをMS-DOSから切り放すことは不可能であるが，INITリクエストではデバイスが能動的にそれ自体の大きさを変更したり，MS-DOSに対する組み込みを中止することができる.

パラメータへのポインタ

デバイスがINITリクエストを受けた時点で，リクエストヘッダのオフセット12H(_bpbptr)内には，このデバイスがロードされる原因となったCONFIG.SYS内の“DEVICE=”文へのポインタが格納されている．このポインタは，“DEVICE=”の直後の文字を指していることから，デバイスはデバイスドライバ名とパラメータを得ることができる.

たとえば，“CONFIG.SYS”内の次のような記述によって“TEST.BIN”がメモリに組み込まれたとする.

```
DEVICE=A:¥TEST.BIN /1234
```

このデバイスに対するINITリクエスト時に“A:¥TEST.BIN /1234”の先頭アドレスを得ることができる．この文字列は，LF(改行)コードで終了している.

デバイスの終了アドレス

デバイスドライバのINITリクエストを実行する処理の多くは，初期化の終了時に不要となることが多い．つまりINITリクエスト処理が終了した以降は二度と実行されないコードができてしまうのである．このようなメモリの無駄使いを防ぐため，INITリクエストの終了時に，デバイスの終了アドレスをMS-DOSに返し，終了アドレス以降のメモリを解放する．具体的には，リクエストヘッダのオフセット0EH(_address)内に解放したいメモリの先頭アドレスをセットする.

初期化失敗時

たとえば，あるデバイスがINITリクエストに応じて初期化処理を行っているとき，制御の対象とするハードウェアが存在しない，などの理由でデバイス全体の存在が無意味になってしまうことがある．このような場合，デバイスの先頭アドレスをデバイス終了アドレスとしてMS-DOSに返す方法が考えられる．しかし，INITリクエストが終わり，MS-DOSに戻った時点でデバイスリンクが書き換えられ，そのデバイスはデバイスリンクチェーン内に組み込まれてしまう．デバイス全体をメモリから消してしまうとデバイスリンクチェーンがそこで破壊され，暴走の原因となる．ERRORビットをセットしてもデバイスは組み込まれてしまう.

初期化失敗時にそのデバイスを完全にメモリから消し，デバイスリンクチェーンに影響を与えないためには，次の手順を実行する必要がある.

リスト5 device.inc

```
        device.inc
        structure of device driver request header.
reqheader       struc
        _length         db      ?
        _unit           db      ?
        _cmd            db      ?
        _status         dw      ?
        _reserve        db      8 dup (?)
        _media          db      ?
        _address        dd      ?
        _count          dw      ?
        _sector         dw      ?
        _volid          dd      ?
reqheader       ends

reqheader2      struc
                        db      13 dup (?)
        _unitnum        db      ?
                        dd      ?
        _bpbptr         dd      ?
        _devnum         db      ?
reqheader2      ends


vector          struc
        _ofs            dw      ?
        _seg            dw      ?
vector          ends

```

リスト 6　test.asm

```
.186
;          test.asm        INIT sample.
;          masm test1; link test;  exe2bin test  によってtest.binを作成する.

RF_INIT              =       0

           include device.inc

code       segment word public
           assume  cs:code, ds:code, es:nothing, ss:code

header          dd      -1
attr            dw      8000h             ;default attribute.
                dw      strategy
                dw      intentry
                db      'TEST    '

ptrsave         vector  <,>               ; request header save.
error_code      db      0

strategy        proc    far
                mov     word ptr cs:ptrsave._ofs, bx
                mov     word ptr cs:ptrsave._seg, es
                ret
strategy        endp

;          device interrupt entry point.

intentry        proc    far
                pushf
                irp     reg,<ax,bx,cx,dx,si,di,ds,es>
                  push  reg
                endm

                mov     cs:error_code, 0
                les     bx, cs:ptrsave
                mov     al, es:[bx].cmd
                cmp     al, RF_INIT
                je      init
                mov     cs:error_code, 3          ; illegal command.
                mov     ah, 80h                   ; error flag.
                jmp     init_x

init:           lds     si, es:[bx]._bpbptr
                cld
                mov     ah, 2             ; put char function.
display:        lodsb
                push    ax
                mov     dl, al
                int     21h
                pop     ax
                cmp     al, 0ah           ; end of parameter ?
                jne     display
                mov     dl, 0dh
                int     21h
;
                mov     ah, 0             ; init complete (no error)
init_x:         mov     es:[bx]._address._seg, cs
                mov     es:[bx]._address._ofs, offset code:header
                mov     es:[bx]._unitnum, 0
                mov     cs:attr, 0

                or      ah, 00000001b             ; set DONE bit.
                mov     al, cs:error_code
                mov     es:[bx]._status, ax       ; set status (init time only)
                irp     reg,<es,ds,di,si,dx,cx,bx,ax>
                  pop   reg
                endm
                popf
                ret
intentry        endp
code            ends
                end
```

①デバイスアトリビュートの内容を0とする．

②リクエストヘッダのオフセット0DH（_unitnum）の内容を0とする．

③デバイス終了アドレス（_address）にデバイスの先頭アドレスを格納する．

そして，INITリクエストを正常終了するのである．この手順の意味は，自分自身を，0台のドライブを制御するブロックデバイスに強制的に変更してしまうことである．MS-DOSは，このようなブロックデバイスをデバイスリンクに組み込まないため，デバイス全体が消去されても問題はない．

INIT処理サンプルプログラム（test.asm）

キャラクタデバイスのINITリクエスト処理における以上のような処理を組み込んだデバイスドライバ"TEST.BIN"を作成した．このデバイスは，INITリクエストを受けると，まずパラメータへのポインタを取得してディスプレイに表示し，次に初期化失敗時の処理を行った後，正常終了する．

実質的には何の役にも立たないが，実際にデバイスドライバを記述する際の参考としていただきたい．

※INITリクエストではDEINSTALLリクエストをサポートしているかどうか，という情報を返す必要がある．これについてはサンプルプログラム2で実現し解説している．

デバイスドライバ
サンプル

サンプル1(test1.asm) 複数のデバイスヘッダをもつデバイスドライバ

このデバイスドライバ（test1.exe）は，1つのプログラム内に複数のデバイスを記述する例であり，とくにデバイスとしての機能は果たさない．"test1.exe"の目的は，以下の3点である．

- EXE形式デバイスドライバの記述を行う．
- 1つのプログラム内に複数のデバイスを記述する．
- デバイスリンクチェーン内への新規デバイスの組み込み機構を知る．

EXE形式のデバイスドライバ

このプログラムはEXE形式を採用しており，

masm test1;
link test1;

リスト 7 test1.asm

```
.186
;               test1.asm       -- multi-header device ---
;
;               masm test1; link test1  によってtest1.exeを作成する.

                include device.inc

code    segment word public
        assume  cs:code, ds:code, es:nothing, ss:code

;       1st. header
header          dw      offset code:header1
                dw      0

                dw      08000h          ;default attribute.
                dw      strategy
                dw      intentry
                db      'TEST1_00'

ptrsave         vector  <,>             ; request header save.
devnumber       db      -1

strategy        proc    far
                mov     cs:devnumber, 0
strategy01:     mov     word ptr cs:ptrsave._ofs, bx
                mov     word ptr cs:ptrsave._seg, es
                ret

strategy1:      mov     cs:devnumber, 1
                jmp short strategy01
strategy        endp

;       2nd. header
header1         dd      -1
                dw      08000h          ;default attribute.
                dw      strategy1
                dw      intentry
                db      'TEST1_01'


;       device interrupt entry point.

intentry        proc    far
                pushf
                irp     reg,<ax,bx,cx,dx,si,di,ds,es>
                  push  reg
                endm
                les     bx, cs:ptrsave
                call    dispmyaddress
                mov     di, offset code:initmsg1
                mov     al, cs:devnumber
                call    bin2hex
                mov     ax, code
                mov     ds, ax
                mov     ah, 9
                mov     dx, offset code:initmsg
                int     21h

                mov     cl, 0
                lds     si, dword ptr cs:header
                call    dispaddress

                mov     cl, 1
                lds     si, dword ptr cs:header1
                call    dispaddress

                mov     cl, 0ffh
                mov     ax, 0000
                mov     ds, ax
                mov     si, 8b38h               ; NUL device.
                lds     si, ds:[si]
                call    dispaddress

exit:           mov     ah, 00000001b           ; set DONE bit.
                mov     al, 0
                mov     es:[bx]._status, ax     ; set status (init time only)
                mov     es:[bx]._address._seg, cs
                mov     es:[bx]._address._ofs, offset code:caution

                irp     reg,<es,ds,di,si,dx,cx,bx,ax>
                  pop   reg
                endm
                popf
                ret

initmsg         db      "init device "
initmsg1        db      "xx", 0dh, 0ah, '$'

intentry        endp

;       information.

amsg            db      "top address = ", '$'
dmsg            db      "device link"
dnum            db      "xx = "
segbuffer       db      "xxxx:"
                db      "xxxx"
crlf            db      0dh, 0ah, '$'


;       convert ds:si into hex string, and store into [segbuffer]
storeaddress    proc    near
                mov     di, offset code:segbuffer

                mov     ax, ds
                xchg    ah, al
                call    bin2hex

                mov     ax, ds
                add     di, 2
                call    bin2hex

                add     di, 3
                mov     ax, si
                xchg    ah, al
                call    bin2hex

                mov     ax, si
                add     di, 2
                call    bin2hex
                ret
storeaddress    endp


dispmyaddress   proc    near
                mov     ax, code
                mov     ds, ax

                mov     dx, offset code:crlf
                mov     ah, 9
                int     21h
                mov     dx, offset code:amsg
                mov     ah, 9
                int     21h
                mov     si, 0
                call    storeaddress
                mov     dx, offset code:segbuffer
                mov     ah, 9
                int     21h
                mov     dx, offset code:crlf
                mov     ah, 9
                int     21h
                ret
dispmyaddress   endp


;       display address.(ds:si) header number = cl.
dispaddress     proc    near
                mov     al, cl
                mov     di, offset code:dnum
                call    bin2hex                 ; store header number.
                call    storeaddress
                push    ds
                push    cs
                pop     ds
                mov     dx, offset code:dmsg
                mov     ah, 9
                int     21h
                pop     ds
                ret
;
dispaddress     endp


;       convert al -> DX. store word ptr cs:[di]
bin2hex         proc    near
                push    bx
                mov     ah, al
                and     al, 0f0h
                shr     al, 4
                mov     bx, offset code:xlattable
                xlat    cs:[bx]
                mov     dl, al                  ; upper nibble.

                mov     al, ah
                and     al, 0fh
                xlat    cs:[bx]
                mov     dh, al                  ; lower nibble.
                mov     cs:[di], dx
                pop     bx
                ret

xlattable       db      "0123456789abcdef"
bin2hex         endp


caution         proc    near
                mov     ax, code
                mov     ds, ax
                mov     dx, offset msg
                mov     ah, 9
                int     21h
                mov     ax, 4c00h
                int     21h

msg             db      'This program is device-driver.', 0dh, 0ah, '$'
caution         endp
code            ends
                end     caution
```

により“test1.exe”を作成し，MS-DOSに組み込む．拡張子に“EXE”をもつプログラムは通常のコマンドと見分けがつかないため，あやまってコマンドラインで実行してしまうおそれがある．そのための対策としてこのプログラムは，通常のコマンドとして実行されたときに，

This program is device-driver.

と画面に表示し，即座に終了するようになっている．具体的には，プログラムエントリアドレスにある“caution”という手続きのみを実行し，プロセス終了ファンクションの実行によって終了する．エントリアドレスの設定は，通常のEXEプログラムと同様に，ソースファイルの末尾にある“end”ステートメントにラベルを与えることによって行う．

EXE形式以外のデバイスドライバの場合，LINK後にEXE2BINコマンドによってメモリイメージに等しいバイナリ形式への変換を行うが，プログラム内でセグメント名(CS, DS, SSなどのセグメントレジスタではなくCODE, DATA, STACKなど)の参照を行っている場合，EXE2BINコマンドはファイル変換を行うことができない．なぜならば，実際にEXEファイルがロードされた後，はじめてそれらのセグメント名に対応するアドレスが決定するからである．実際のセグメント値の決定を行うための情報は，EXEヘッダ内のリロケーションテーブルに用意されている．

“test1.exe”ではCODEセグメントの参照を行っているが，実際にデバイスとして組み込まれた段階で正しいセグメント値に置き換えられている．つまり，MS-DOSによるデバイスドライバの読み込み時にも，EXEヘッダの解析や参照の解決が行われていることがわかる（INT 21Hファンクション4B03Hのロードオーバーレイファンクションに近い機能が利用されていると考えられる）．

●●● 複数のデバイスヘッダ

1つのファイル内に複数のデバイスを記述することは，1つのプログラム内に複数のデバイスヘッダを用意することに等しい．通常の場合（1プログラムに1デバイスの場合），デバイスリンクの初期値は－1とするが，複数のデバイスを備えているファイルの場合にはそのプログラム内でのリンクをあらかじめ記述しておく．そして，そのプログラム内での最終デバイスのデバイスリンクにのみ－1を記述する．デバイスリンクは，セグメントとオフセットをもつ4バイトのフィールドであるが，1つのプログラム内でのリンクについてはオフセットのみでよい．セグメント値はINITリクエストが正常に終了した際に，MS-DOSによって補われる．

このプログラムには2つのデバイスヘッダ（header，header1）が用意されており，それぞれ“TEST1_00”，“TEST1_01”というデバイス名をもっている（以下，簡単のためにそれぞれを「_00デバイス」，「_01デバイス」と表記する）．headerのデバイスリンクの初期値は“0000:(header1のオフセット)”，header1側は“FFFF:FFFF”としている．

デバイスヘッダ以外のデバイス割り込みルーチン（intentry）やリクエストヘッダの保存領域(ptrsave)は，2つのデバイスが共用している．2つのデバイスストラテジルーチン（strategy，strategy1）では，共用ルーチン内でリクエストを受けたデバイスを区別するためにそれぞれのデバイスに対応する番号（_00デバイスは0，_01デバイスは1）をメモリ（devnumber）内に書き込んでいる．

MS-DOSは一度に1つのデバイスに対してのみデバイスリクエストを発行するため，デバイスストラテジルーチンで保存されたリクエストヘッダのポインタは，対応するデバイス割り込みルーチンによって正しく参照される．複数のデバイスヘッダをもつ実用的なデバイスドライバを記述する際にも，このプログラムのようにルーチンの共用は可能であろう．

●●● プログラムの動作説明

“test1.exe”は，デバイスリクエストを受けるといくつかの内部情報をコンソールに表示する．これは，デバイスリンクの初期化順序を解説するために用意した機能である．このデバイスドライバは，すべてのリクエストに対してリクエストヘッダのステータスフ

図3 コンソールへの情報出力時の書式

top address = segment:offset	←	“test1.exe”が配置されているアドレス．
init device ##	←	リクエストされたデバイス番号（devnumber）
device link00 = segmen:offset	←	2つのデバイスヘッダそれぞれが保持している
device link01 = segmen:offset	←	デバイスリンクの内容．
device linkff = segment:offset	←	NULデバイスのデバイスヘッダ

ィールド内に正常終了ステータスを格納し，アドレスフィールド内にデバイスドライバとしてメモリに常駐する際の最終アドレスを格納する．つまり，デバイスとしては何1つ具体的に動作せず，MS-DOSに対してつねに正常にリクエストが処理された，という情報のみを返すようにしている．実際のところ，これら2つのデバイスに対し，MS-DOSから見て意味のあるリクエストはINITのみである．

図4に，組み込み時に表示される情報と，組み込み前と組み込み後それぞれのデバイスリンクチェーンを示す．デバイスリンクチェーンは，NULデバイスのデバイスリンクに書かれているアドレス（最後に組み込まれたデバイスのアドレスが格納されている）を開始点として，順次リンクをたどっていくことで得られる．

"test1.exe"は，デバイスリクエストを受けると図3のような書式でコンソールに情報を出力する．

図4　test1.exeの組み込み

a）組み込み時に表示される内容

```
top address = 205b:0000       ← 実際に配置されたアドレス

init device 00                ← _00デバイスが先に初期化される.
device link00 = 0000:0030
device link01 = ffff:ffff     ← _00, _01ともにデバイスリンクは初期値のまま
device linkff = 1072:0036     ← NULはCONを指している（図b参照）

top address = 205b:0000

init device 01                ← 次に_01デバイスが初期化される.
device link00 = 1072:0036     ← 直前のNULのデバイスリンクが格納された.
device link01 = ffff:ffff     ← _01のデバイスリンクは初期値のまま
device linkff = 205b:0000     ← _00デバイスのヘッダアドレスが格納された.
```

b）test1.exeの組み込み前のデバイスリンクチェーン

デバイスヘッダのアドレス	デバイスリンクの内容	デバイス名
0000:8b38	1072:0036	NUL
1072:0036	1072:0024	CON
1072:0024	1072:0000	MS$KANJI
1072:0000	…	AS$VJE
	ffff:ffff	

b'）_00デバイスのみを組み込んだ時点でのデバイスリンクチェーン

デバイスヘッダのアドレス	デバイスリンクの内容	デバイス名
0000:8b38	205b:0000	NUL
205b:0000	1072:0036	TEST1_00
1072:0036	1072:0024	CON
1072:0024	1072:0000	MS$KANJI
1072:0000	…	AS$VJE
	…	

●●● デバイスリンクチェーンへの登録

図4-aは，"test1.exe"をMS-DOSに組み込む際に表示された内容である．ここから，組み込み時にはこのプログラム内の2つのデバイスに対して順次デバイスリクエストが発行されていることがわかる．デバイスリンクの内容に注目すると，最初のリクエスト時では初期値のままであるが，2度目のリクエストを受けた時点では_00デバイスのデバイスリンクが変化し"1072:0036"という値になっている．そしてNULデバイスのデバイスリンク内容も変化し，"205b:0000"となっている．これは，INITリクエストが正常に終了したことによって，_00デバイスがデバイスリンクチェーンの最上位に組み込まれたことを意味する．

図4-bは"test1.exe"を組み込む直前のデバイスリンクチェーンである．ここから，_00デバイスに与えられた"1072:0036"というアドレスは，"test1.exe"の1つ前にデバイスリンクチェーンに組み込まれていた（つまり，デバイスリンクチェーンの最上位にあった）CONデバイスのデバイスヘッダを指していることがわかる．そして，その時点でのNULデバイスのデバイスリンクにも同じ"1072:0036"というアドレスが格納されている．図4-a，図4-bおよび4-b'からわかることをまとめると以下のようになる．

1. MS-DOSは，新たにメモリに配置したデバイスの先頭にあるデバイスヘッダ情報を基にINITリクエストを発行する．
2. この段階ではデバイスリンクチェーンは変化しない．
3. デバイスがINITリクエストに対して正常終了ステータスを返してはじめてデバイスリンクの置き換えが行われる．
4. NULデバイスのデバイスリンクには，つねに最後に登録されたデバイスの先頭アドレス（デバイスヘッダのアドレス）が格納される．

●●● 同一プログラム内の複数デバイスに対する初期化シーケンス

単一のデバイスヘッダしかもたないプログラムの場合，これまでの解説で新規デバイスの組み込みは終了する．しかし"test1.exe"は，形式上2つのデバイスを抱えており，_00デバイスの組み込みだけでは話が終わらない．図5に"test1.exe"の組み込みが終了し

図 5

"test1.exe"の組み込みが終了した直後のデバイスリンクチェーン

組み込み後

デバイスヘッダのアドレス	デバイスリンクの内容	デバイス名
0000:8b38	205b:0030	NUL
205b:0030	205b:0000	TEST1_01 (組み込み前は, ffff:ffff)
205b:0000	1072:0036	TEST1_00 (組み込み前は, 0000:0030)
1072:0036	1072:0024	CON
1072:0024	1072:0000	MS$KANJI
1072:0000		AS$VJE
	...	

た直後のデバイスリンクチェーンを示す．図4-bと比較すると，新たに_01デバイスが_00デバイスの上位に組み込まれていることがわかる．

複数のデバイスをもつプログラムでは，デバイスリンクの初期値によって複数のデバイスヘッダをつなぐリストを形成することになっている．そのリストの先頭に位置するデバイスヘッダは，必ずプログラムの先頭アドレスに位置していなければならず，リストの末尾に位置するデバイスのデバイスリンクは－１とする必要がある．

MS-DOSは，このリストの順序にしたがってINITリクエストを発行し，デバイスリンクチェーンへの組み込みを行う．そしてリストの末尾にあるデバイスの初期化が終了した時点で１つのプログラムの組み込みが終了することになる．つねに，最後に組み込まれたデバイスが，デバイスリンクチェーンの最上位に位置するため，リストの末尾に近いデバイスほどデバイスリンクチェーンでは上位に組み込まれることに注意していただきたい．

以上から，複数のデバイスヘッダをもつデバイスドライバを組み込む際のMS-DOSの処理の流れは以下のように推測できる．

①新規組み込みデバイスドライバ用のメモリブロックを作成する（セグメント値NEWが返る）．

②デバイスドライバをセグメントNEWの先頭から読み込む．

③内容がセグメントNEW，オフセット０のアドレス変数HEADを作成する．

④HEADが指すデバイスヘッダについて，INITリクエストを行う．

⑤終了ステータスを解析し，アドレスフィールドの内容にしたがってメモリブロックのサイズを変更する．

⑥新規組み込みデバイス内デバイスリンクの初期値（TMP）を一時的に保存．

⑦NULのデバイスリンクの内容を新規組み込みデバイスのデバイスリンクへ複写．

⑧HEADをNULのデバイスリンクへ複写．

⑨TMPがFFFF:FFFFならば，このデバイスドライバの組み込みはすべて終了．

⑩TMPがFFFF:FFFFでない場合，HEADにTMPを複写し，HEADのセグメント部にNEWを複写する．

⑪処理④に分岐する．

デバイスドライバの組み込みに関する部分については，コマンドレベルでデバイスドライバの組み込みを行うための"ADDDRV.EXE"も同様な処理を行っていると考えられる．

サンプル２（test2.asm）ADDDRV/DELDRVコマンド用DEINSTALLリクエスト

PC-9800シリーズ用などのMS-DOSシステムディスクには，コマンドラインからキャラクタデバイスの組み込み／切り放しを行うための"ADDDRV.EXE"，"DELDRV.EXE"コマンドが用意されており，その動作は興味をひく．そして，これらのコマンドに対応したDEINSTALLリクエストが用意されている（PC-9801用MS-DOS Ver.3.3C以外では確認していない）．ここでは，このデバイスリクエスト用の処理を内蔵したデバイスドライバ"TEST2.EXE"を紹介する．

このプログラムは，サンプル１の"TEST1.ASM"をさらに単純化したものであり，INITリクエストとDEINSTALLリクエスト用の処理のみを備えている．

ADDDRV／DELDRVのデバイス組み込み／切り放し機構（DEINSTALLなしの場合）

"ADDDRV.EXE"によってデバイスの組み込みを行う際には，組み込むデバイスを記述したデバイス定義ファイルが必要である．この例では，リスト９のようなデバイス定義ファイル"TEST2.INI"を作成した．

実際の組み込みを行う際には，コマンドラインに"ADDDRV TEST2.INI"と入力する．

"ADDDRV.EXE"のEXEヘッダ

の内容を見ると，このプログラムがメモリにロードされたあとに必要とするパラグラフ数の最大最小値（maxalloc と minalloc）は共に“0”となっており，“/HIGH”オプションつきでリンクされたことがわかる．つまり“ADDDRV.EXE”は，MS-DOSが使用可能なメモリ内の，できるだけ高いアドレスにロードされるように作られている．したがって，“ADDDRV.EXE”がロードされ実行を開始した時点ではメモリの中ほどに大きなフリーエリアがある．

“ADDDRV.EXE”はパラメータとして与えられたデバイス定義ファイルを1行ずつ読み込み，指定されているデバイスの組み込みを実行する．デバイスドライバのメモリへの読み込みと配置は，INT 21Hファンクション4B03Hのロードオーバーレイファンクションによって行われていると考えられる．このシステムコールは指定されたセグメントに指定されたファイルを読み込むものであり，同時にリロケーション情報の解析や参照の解決を行う．

メモリに読み込まれたデバイスに対し，“ADDDRV.EXE”はMS-DOSと同様にINITリクエストを発行し，メモリブロックの調整（INITリクエストの戻り値のデバイス最終アドレスに基づいて，デバイスに割り当てられたメモリブロックの大きさを調整する），サンプル1で解説したデバイスリンクチェーンへの組み込みを行う．MS-DOSと異なる点は，デバイスが組み込まれる直前の割り込みベクタテーブルの内容も同時に保存されることである．

“DELDRV.EXE”が起動すると，“ADDDRV.EXE”によって組み込まれたメモリブロックが存在するかどうかを調査する．存在していない場合

リスト 8　test2.asm

```
.186
;       test2.asm       DEINSTALL sample.
;
;       masm test1; link test2  によってtest2.exeを作成する.
;
        include device.inc

RF_INIT         =       0
RF_DEINST       =       20

code    segment word public
        assume  cs:code, ds:code, es:nothing, ss:code

header          dd      -1
                dw      08000h          ;default attribute.
                dw      strategy
                dw      intentry
                db      'TEST2_00'

ptrsave         vector  <,>             ; request header save.

strategy        proc    far
                mov     word ptr cs:ptrsave._ofs, bx
                mov     word ptr cs:ptrsave._seg, es
                ret
strategy        endp

;       device interrupt entry point.

intentry        proc    far
                pushf
                irp     reg,<ax,bx,cx,dx,si,di,ds,es>
                  push  reg
                endm
                les     bx, cs:ptrsave
                mov     al, es:[bx]._cmd
                cmp     al, RF_INIT
                jne     intent1
;       init request.
                not     es:[bx]._bpbptr._ofs    ; DEINSTALL  support
                not     es:[bx]._bpbptr._seg
                mov     es:[bx]._address._seg, cs
                mov     es:[bx]._address._ofs, offset code:caution
                jmp short exit

;       DEINSTALL support.

dimsg           db      "DEINSTALL requested", 0dh, 0ah, '$'

intent1:        cmp     al, RF_DEINST
                jne     exit

                mov     dx, offset code:dimsg
                push    cs
                pop     ds
                mov     ah, 9
                int     21h
exit:           mov     ah, 00000001b           ; set DONE bit.
                mov     al, 0
                mov     es:[bx]._status, ax     ; set status (init time only)

                irp     reg,<es,ds,di,si,dx,cx,bx,ax>
                  pop   reg
                endm
                popf
                ret

intentry        endp
caution         proc    near
                mov     ax, code
                mov     ds, ax
                mov     dx, offset msg
                mov     ah, 9
                int     21h
                mov     ax, 4c00h
                int     21h

msg             db      'This program is device-driver.', 0dh, 0ah, '$'
caution         endp
code            ends
                end     caution
```

にはその旨を表示して終了する．存在している場合，メモリ中に保存されている割り込みベクタテーブルを復帰し，デバイスリンクチェーンを“ADDDRV.EXE”実行前の状態に戻すことにより，実質的にデバイスをシステムから切り放す．最後にデバイスが常駐していたメモリブロックを解放し処理を終了する．

割り込みベクタテーブルの保存と復帰は，組み込まれたデバイスが割り込みベクタの書き換えを行ったとしても，切り放し時には以前の状態に復帰しておこう，という配慮と考えられる．しかしながら，外部からのハードウェア割り込みを取り扱うデバイスに関しては，この処理はまったく意味をもたない．なぜならば，割り込み要因を取り除かないかぎり，正しい割り込みハンドラを失った割り込みがあやまって発生し得るからである．

リスト 9　デバイス定義ファイル“TEST2.INI”の内容

```
DEVICE=TEST2.EXE
```

図 6　“TEST2.EXE”の実行

```
A> adddrv test2.ini
Microsoft (R) Character Device driver install and deinstall  Version 1.01
Copyright (C) Microsoft Corp 1987. All rights reserved.

A> deldrv
DEINSTALL requested

A>
```

●●● DEINSTALL リクエストとその使い方

DEINSTALL リクエストは，このような場合に対する配慮である．INIT リクエストがデバイスの初期化処理の実行要求とすれば，DEINSTALL リクエストは正常終了処理の実行要求である．DEINSTALL リクエストを受けたデバイスは，MS-DOSの動作環境の復元や外部割り込みの停止処理など，必要に応じてデバイス内部の状態を「安全な状態」に戻し，処理を終了する．

“CONFIG.SYS”内の“DEVICE=”文を介して MS-DOS に組み込まれたデバイスについては“DELDRV.EXE”などのようなコマンドで切り放すことはできないため，MS-DOSが直接 DEINSTALL リクエストを発行することはない．DEINSTALL リクエストはあくまでも“ADDDRV.EXE”,“DELDRV.EXE”のために用意された特殊なリクエストである．具体的には，あるデバイスが“ADDDRV.EXE”による組み込み時に INIT コールの戻り値として「DEINSTALL リクエストをサポートしている」という情報を返したとき，次に実行される“DELDRV.EXE”は，このデバイスに対して DEINSTALL 処理を要求し，その処理の終了後に切り放しを行う．

DEINSTALL をサポートするデバイスは，INIT 処理においてリクエストヘッダ内の BPB ポインタフィールド (_bpbptr) 内の全ビットを反転する．これによって“ADDDRV.EXE”は，このデバイスが DEINSTALL 処理をサポートしていること知る．

BPB ポインタフィールドはキャラクタデバイスでは使用されていないため，この機能をサポートしたデバイスを“ADDDRV.EXE”以外のツールや“CONFIG.SYS”経由で組み込んだとしても副作用が起こるおそれはない．

●●● プログラムの説明

“TEST2.ASM”は“TEST1.ASM”のデバイス数を1つとし，情報表示などを取り除いた上で DEINSTALL 処理に関する機能を追加したプログラムである．“TEST1.ASM”と同様に EXE 形式として使用する．このプログラムを組み込むと，図6のように表示され，DEINSTALL リクエストを正しく受け取っていることがわかる．

DEVICE DRIVER

特集
最新「デバイスドライバ」研究

虎岩　登



MS-DOSデバイスドライバの書き方

プログラムの基本構造は共通

MS-DOS デバイスドライバのプログラミングは，通常の DOS プログラミングよりも難しいといわれている．その理由として，MS-DOS とのインターフェースをとるために，独特のデータ構造やコマンド形式が使われること，デバッグが難しいことなどがあげられる．
だが，データ構造やコマンド形式が決まっている以上，デバイスドライバのプログラムはどれもよく似た構造になるはずだ．構造さえ理解できれば，デバイスドライバのプログラミングはそれほど難しいものではないといえる．ここでは，キャラクタ型デバイスとブロック型デバイスのそれぞれについて，デバイスドライバプログラムのひな型を提供し，その内容を解説する．あわせて，実際のプログラミングに関するポイントを紹介する．

プログラミングの前に

MS-DOSのデバイスドライバプログラミングについて，実際のソースプログラムをあげて解説せよとのことだが，プログラムを見る前に，前提として知っておいてほしいことがいくつかある．

デバイスドライバは作りにくい

昨今は，デバイスドライバをCで書こうという人もいるらしい．だが，Cで全部を書くわけにはいかない．どうしても，DOSとのインターフェース部分はアセンブラが必要になる．おまけに，Cの標準ライブラリを使おうと思っても，いろいろな制限――大量のスタックを利用するような関数は使えないとか，ファンクションリクエストINT 21Hを呼び出すものは使えないとか――があるので，事実上かなり難しい．ふつうに書くには，アセンブラを使うということになる．そう，アセンブラだ．なかにはこれだけで億劫になる人もいるだろう（本誌の読者では少ないのかもしれないが）．しかも，デバイスドライバはDOSから呼ばれることを前提としたプログラムだ．DOSの規約を完全に満たさなければいけない．

デバイスドライバのデバッグは難しい

デバイスドライバをデバッグ可能なデバッガというのも，いまは存在する．でも，それはデバッグ不可能だったのが可能になっただけというべきで，依然として通常のプログラムよりもデバッグはめんどうだ．おまけにデバイスドライバは，ちょっとまちがうと暴走しやすい．アセンブラで書くから，わずかなミスが命取りだ．組み込んでテストしようにも，いちいちリセットしなければならない．キャラクタデバイスならADDDRV・DELDRVコマンドを使って組み込むという手もあるが，イニシャライズにバグがあればADD-DRVしたとたんに暴走してしまうこともある．

ほんとうにデバイスドライバが必要なのか？

デバイスドライバというのは何かにつけてめんどうだ．めんどうなものをなぜ作ろうとするのかといえば，それなりに便利な点があるからだ．RAMディスクやSCSIのディスクドライバのようなブロック型のデバイスドライバについては，デバイスドライバ形式でないと意味がないから仕方がない．だが，よほどマイナーな機種でない限りは，買ったほうが安くて確実である．

キャラクタ型デバイスだったら，話がちょっと違ってくる．たとえばプリンタ．特殊なプリンタで印字したいとか，特別な修飾機能を使って印字したいというような場合，たしかにデバイスドライバの形式にすれば，DOS上のさまざまなプログラムの出力を直接印字できるようになるかもしれない．これは便利だ．ただし，デバイスドライバだと作成やデバッグがめんどうだったり，メモリ上に常駐してしまうのでアプリケーションから使えるメモリが減ったりする．

もしも，デバイスドライバではなくて，指定されたファイルを印字するプログラムというのを作ったとしたらどうなるか？　このプログラムは与えられたファイルしか印字できない．ほかのプログラムの出力を印刷したいと思っても，一度リダイレクトしてファイルに落とさなければ印刷できないことになる．でも，このプログラムはCで書こうがBASICで書こうがかまわない．最近はやりの統合環境から一歩も出ないで作成からデバッグまでできる．果たしてどちらが得（あるいは楽）だろう？

自分が作るプログラムの中から利用するだけならばサブルーチンか，ライブラリという形式で入出力を司る部分を独立させておいたほうがよい．すでに提供されているプログラムでも利用したいというときに，はじめてデバイスドライバとして作成すればいいのだ．

気をつけておきたいポイント

最初にデバイスドライバを書くときに失敗しやすい問題として，ソースファイルの名前がある．デバイスの名前がFOOだと，FOO.ASMなんていう名前にしたくなるが，デバイスとして組み込んだとたんに悲劇が起こる．FOO.ASMは読むことも書くこともできなくなってしまうのだ．MS-DOSではデバイス名とファイル名とではデバイス名のほうが優先順位が高いので，type foo.asmとやっても，FOOというデバイスから入力したデータを画面に出力しようとしてしまう．ソースファイルの名前は，必ずデバイス名と異なったものをつけよう．

NEC PC-98シリーズ一覧
PC-98XL（1986/10発表）●CPU:V30(8/10MHz)/80286(8/10MHz)●メインメモリ：1.0MB●スロット数：4 ●備考：XA後継機，ハイレゾ，model4は20MBHDD内蔵

サンプルプログラム解説

では，実際にキャラクタ型とブロック型のデバイスドライバのサンプルプログラムを紹介する．両方ともDOSとのやりとりの部分を参考にできるように，余分なところは省いた，ほとんど骨だけのプログラムである．

●プログラムの概要

キャラクタ型デバイスドライバの例としては，何もしないプリンタドライバを取りあげた．これは，送られてきたデータをプリンタに出力するだけのデバイスドライバだ．tabをスペースに変換したり，漢字コードを変換するような処理はいっさいやっていない．したがって，漢字を印字させると変なグラフィックキャラクタが出たりするはずだ．このままでは実用性はないが，これをベースに機能をつけ加えていってほしい．

ブロック型デバイスドライバの例としては，RAMディスクをとりあげた．これもメインメモリから32KBをディスクのように使おうというドライバだが，その32KBも最初から確保してある分を使うというものだ．こちらもこのままでは実用性はない．

●devdr.inc

devdr.incは，2つのプログラムで共通に使われるヘッダファイルだ．これは，各コマンドパケットのオフセット値を構造体定義を利用して定義している．名前のほうが覚えやすいからだ．

●●● far_ptr

FARポインタのデータ型の定義．

●●● req_header

MSDOS.SYSとデバイスドライバとのやりとりで使用されるコマンドパケットの共通部分，リクエストヘッダの構造体．

●●● init_cmd_form

コマンドコードが0の場合，INITコマンドパケットの構造体．

●●● media_chk_form

コマンドコードが1の場合，MEDIA CHECKコマンドパケットの構造体．MEDIA CHECKコマンドは，ブロック型デバイスでのみ有効．rdrv.asmで使用する．

●●● build_bpb_form

コマンドコードが2の場合，BUILD BPBコマンドパケットの構造体．BUILD BPBコマンドは，ブロック型デバイスでのみ有効．rdrv.asmで使用する．

●●● rw_cmd_form

コマンドコードが4，8，9の場合，READまたはWRITEのコマンドパケットの構造体．rdrv.asm，prin2.asmの両方で使用する．

●●● char_dev_header

キャラクタ型デバイスドライバ用のデバイスヘッダを定義する構造体．prin2.asmで使用する．

●●● blk_dev_header

ブロック型デバイスドライバ用のデバイスヘッダを定義する構造体．

リスト1 DEVDR.INC

```
;
;               struct definition
;
far_ptr         struc                           ;FARポインタのデータ型
fp_off          dw      ?
fp_seg          dw      ?
far_ptr         ends
;
req_header      struc                           ;リクエストヘッダの型
req_len         db      ?
                db      ?
req_code        db      ?
req_stat        dw      ?
                db      8 dup (?)
req_header      ends
;
init_cmd_form   struc                           ;初期化コマンドのパケット定義
                req_header      <>
init_unit       db      ?
init_end_addr   far_ptr <>
init_bpb_addr   far_ptr <>
init_dev_no     db      ?
init_cmd_form   ends
;
media_chk_form  struc                           ;メディアチェックコマンドのパケット
                req_header      <>
mchk_media_id   db      ?
mchk_val        db      ?
mchk_vol_id     dd      ?
media_chk_form  ends
;
build_bpb_form  struc                           ;BUILD BPBコマンドのパケット
                req_header      <>
bbpb_media_id   db      ?
bbpb_tfr_addr   far_ptr <>
bbpb_bpbptr     far_ptr <>
build_bpb_form  ends
;
rw_cmd_form     struc                           ;Read/Writeコマンド関連のパケット
                req_header      <>
rw_media_id     db      ?
rw_tfr_addr     dd      ?
rw_count        dw      ?
rw_sec_no       dw      ?
rw_vol_id       dd      ?
rw_cmd_form     ends
;
char_dev_header struc                   ;キャラクタ型デバイスドライバのヘッダ定義
dev_link        far_ptr <-1, -1>
dev_type        dw      8000h
dev_strategy    dw      ?
dev_entry       dw      ?
dev_name        db      '        '              ;8つのスペース
char_dev_header ends

blk_dev_header  struc                   ;ブロック型デバイスドライバのヘッダ定義
                far_ptr <-1, -1>
                dw      2000h
                dw      ?
                dw      ?
dev_count       db      1
                db      '       '               ;7つのスペース
blk_dev_header  ends
```

表1　コマンド種別

コマンドコード	ファンクション	キャラクタデバイス	ブロックデバイス
0	INIT	○	○
1	MEDIA CHECK	—	○
2	BUILD BPB	—	○
3	IOCTL INPUT	*1	*1
4	INPUT	○	○
5	NON-DESTRUCTIVE INPUT NO WAIT	○	—
6	INPUT STATUS	○	—
7	INPUT FLUSH	○	—
8	OUTPUT	○	○
9	OUTPUT WITH VERIFY	○	○
10	OUTPUT STATUS	○	—
11	OUTPUT FLUSH	○	—
12	IOCTL OUTPUT	*1	*1
13	DEVICE OPEN	*2	*2
14	DEVICE CLOSE	*2	*2
15	REMOVABLE MEDIA	—	*2
16	OUTPUT UNTIL BUSY	*3	—
19	Generic IOCTL	*4	*4
20	DEINSTALL	○	—
23	Get Drive Map	—	*4
24	Set Drive Map	—	*4

*1　デバイスドライバがIOCTL機能をもつ場合のみ
*2　デバイスドライバがOPEN/CLOSE/RM機能をもつ場合のみ
*3　ビット13がセットされている場合のみ
*4　V3.3ビット（デバイスヘッダの属性のビット6）が1のデバイスのみ

図1
デバイスヘッダの構成

2word	次のデバイスドライバへのポインタ 1つのファイルに複数のデバイスドライバが含まれる場合に続くデバイスヘッダをポイントする。 最終デバイスドライバは-1にしておく。通常は一つのファイルにデバイスドライバは一つしか含まれないことが多いが，その場合も-1にする。
1word	アトリビュート bit15　ブロックデバイスのとき0 　　　　キャラクタデバイスのとき1 bit14　IOCTL ファンクションサポートの有無（1/0） bit13　Output Until Busy（char device） 　　　　Non Fat ID（Block device） bit12　0（reserved） bit11　OPEN/CLOSE/RM ファンクションサポートの有無（1/0） bit10 ｜　　0（reserved） bit7 bit6　V3.3bit，MS-DOS V3.3で追加されたファンクションサポートの有無 bit5　0（reserved） bit4　特殊デバイス（char device only） bit3　カレント clock デバイス（char device only） bit2　カレント NUL デバイス（char device only） bit1　カレント標準出力デバイス（char device only） bit0　カレント標準入力デバイス（char device only）
1word	ストラテジエントリへのポインタ
1word	インタラプトエントリへのポインタ
8Byte	デバイスの名称(char device) サポートしているユニット数（先頭バイトのみ　Block device）

rdrv.asmで使用する。

● prin2.asm

prin2.asmは，プリンタドライバのソースプログラムだ。最初にheaderでデバイスヘッダを定義している。デバイス名は「PRN2」。デバイスドライバの入り口は，ストラテジエントリとインタラプトエントリの2つがある。ストラテジエントリstrategyでは，ただMS-DOSから渡されたポインタをワークエリアにセーブしておくだけでよい。

実際に作業をするのは，インタラプトエントリint_entryが呼ばれたときだ。リクエストコードの内容にしたがってcmd_tableを使ってジャンプする。デバイスドライバでサポートしなければならないコマンドの種類は，バージョンが新しくなるごとに増え続けているが，ふつうの処理には昔ながらの13種類のコマンドでたりてしまう（表1）。デバイス属性ワードで，新しいコマンドをサポートするかどうかを示す決まりになっている（図1）。しかも，キャラクタ型やブロック型のどちらかしか使わないコマンドがあるので，実際に作成すべき処理ルーチンというのはたいした数ではない。

プリンタデバイスだと，出力しか行わないので，0，8，9，10番のコマンド処理だけでたりる。0がデバイスドライバの初期化，8が出力，9はベリファイ付きの出力だが，プリンタだからふつうの出力と同じことをすればよい。10は出力のステータスを返す。バッファリングを行う場合には，さらに11のoutput flushに対応すればよい。

文字出力はpr_outで行われる。コマンドパケットから出力文字数とポイン

リスト 2

PRIN2.ASM

```
;
;       例題プリンタデバイスドライバ
;
                include devdr.inc
;
;       device driver body
;
code            segment word public     'code'
                assume  cs:code, ds:code, es:code, ss:code
;
;       デバイスヘッダの定義
;
header          char_dev_header <,, offset strategy, offset int_entry, 'PRN2'>
;
cmd_pack_ptr    far_ptr <>
;
;       ジャンプテーブル
;
cmd_table       dw      offset init     ;init
                dw      offset cmd_err  ;media check
                dw      offset cmd_err  ;build BPB
                dw      offset cmd_err  ;IOCTL input
                dw      offset cmd_err  ;input
                dw      offset cmd_err  ;non-destructive input no wait
                dw      offset cmd_err  ;input status
                dw      offset cmd_err  ;input flush
                dw      offset pr_out   ;output
                dw      offset pr_out   ;output with verify
                dw      offset pr_stat  ;output status
                dw      offset exit     ;output flush
;
;
;       ストラテジエントリの処理
;
strategy        proc    far
                mov     cs:[cmd_pack_ptr.fp_off], bx
                mov     cs:[cmd_pack_ptr.fp_seg], es
                ret
strategy        endp
;
;
;       インタラプトエントリの処理
;
int_entry       proc    far
                push    ax              ;レジスタをすべて保存
                push    bx
                push    cx
                push    dx
                push    si
                push    di
                push    bp
                push    ds
                push    es
                lds     si, cs:[cmd_pack_ptr]
                mov     bl, [si.req_code]
                cmp     bl, 11
                ja      cmd_err
                xor     bh, bh
                shl     bx, 1
                mov     ax, offset cmd_table
                add     bx, ax
                mov     ax, cs
                mov     ds, ax
                jmp     [bx]            ;コマンドに応じて分岐
;
cmd_err:        mov     ax, 8103h
                jmp     exit1
;
exit:           mov     ah, 01h         ;正常終了の戻り口
exit1:          lds     di, cs:[cmd_pack_ptr]
                mov     [di.req_stat], ax
                pop     es              ;レジスタを復帰する
                pop     ds
                pop     bp
                pop     di
                pop     si
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                ret
;
;
;       プリンタへ文字出力を行う
;
pr_out:         lds     si, cs:[cmd_pack_ptr]   ;コマンドパケットから
                mov     cx, [si.rw_count]       ;出力文字数と
                lds     si, [si.rw_tfr_addr]    ;文字の格納場所を取り出す
                cld
pr_out1:        lodsb                           ;指定された数だけ出力する
                mov     ah, 11h
                int     1ah                     ;エラーコードがahに戻る
                loop    pr_out1
                jmp     exit1                   ;終了
;
;
;       プリンタステータスの取得
;
pr_stat:        mov     ah, 12h
                int     1ah
                and     ah, 01h
                jnz     exit
                mov     ah, 03h
                jmp     exit1
;
int_entry       endp
;
init:           mov     ax, cs
                mov     ds, ax
                mov     dx, offset init_msg     ;メッセージ出力
                mov     ah, 09h
                int     21h
;
                mov     ah, 10h                 ;プリンタの初期化
                int     1ah
;
                lds     bx, [cmd_pack_ptr]      ;MS-DOSへの戻り値をセット
                mov     [bx.init_unit], 1
                mov     [bx.init_end_addr.fp_off], offset init
                mov     [bx.init_end_addr.fp_seg], cs
                jmp     exit
;
init_msg        db      0dh, 0ah, "Sample printer driver.", 0dh, 0ah, "$"
;
code            ends
                end
```

タを取り出して，順番に出力するだけだ．

初期化操作では，組み込み時のメッセージを出力した後，デバイスドライバの常駐部の最後のアドレス＋1を返すのを忘れないように．つまり，INITルーチンは一度しか実行されないので，これを最後において，INITルーチンの先頭をデバイスドライバの最終アドレスとすれば，それだけメモリの節約になるのである．

ここではオプション指定など行っていないので無視しているが，このファンクションが呼び出されたときinit bpb_addrには，config.sysに書かれている「device =」の次の文字列へのポインタが入っている．だから，これを解析してオプションに応じたデバイスの初期設定を行うことも可能である．

● rdrv.asm

rdrv.asmは，RAMディスクドライバのソースプログラムである．対応するコマンドが，プリンタドライバとちょっと違うが，基本的な構造に違いはない．ストラテジエントリ strategy で，MS-DOSから渡されたポインタをセーブし，実際の処理はインタラプトエントリ int_entry procが呼ばれたときに行う．リクエストコードの内容にしたがって，cmd_tableを使ってジャンプするのも同じである．

このRAMディスクドライバでは，0，1，2，4，8，9番のコマンド処理を行う．0は初期化．1はディスクの交換チェック．ここでは常に1(交換されていない)を返す．2はBPBの取り出し．常に同じBPBアドレスを返している．4はディスク読み込み，8，9はディスクへの書き込みである．

また，キャラクタ型の場合にはデバイスヘッダに8バイトでデバイスの名前を書くとことになっているが，ブロック型デバイスでは，ここには先頭バイトにサポートするデバイス数を入れて残りは使わない．だが，残りの7バイトに名前を書き込んでおくと，デバイスドライバ表示用のユーティリティによってはその名前を表示してくれる．

また，1セクタ無駄になるようだが，ブートセクタに相当する先頭セクタにメディアディスクリプタテーブルを書き込んでおくと，一部ディスクユーティリティなどでそこを読み取ったりするものがあるのでいいことがあるかも

リスト3　RDEV.ASM

```
;
;                例題RAMディスクドライバ
;
;
                 include devdr.inc
;
code             segment word public     'code'
                 assume  cs:code, ds:code, es:code, ss:code
;
SEC_SIZ          equ     512               ;1セクタのサイズ
SEC_MAX          equ     64                ;ドライブ当たりのセクタ数
MEDIA_ID         equ     0ffh              ;メディアディスクリプタ
;
;        デバイスヘッダの定義
;
header           blk_dev_header  <,,strategy, int_entry, 1, 'SAMPLE'>
;
cmd_pack_ptr     far_ptr <>
;
;
;        ジャンプテーブル
;
cmd_table        dw      init              ;init
                 dw      med_chk           ;media check
                 dw      get_bpb           ;build BPB
                 dw      cmd_err           ;IOCTL input
                 dw      dsk_read          ;input
                 dw      cmd_err           ;non-destructive input no wait
                 dw      cmd_err           ;input status
                 dw      cmd_err           ;input flush
                 dw      dsk_write         ;output
                 dw      dsk_write         ;output with verify
;
bpb_tbl          dw      offset BPB
;
;
;        ストラテジエントリの処理
;
strategy         proc    far
                 mov     cs:[cmd_pack_ptr.fp_off], bx
                 mov     cs:[cmd_pack_ptr.fp_seg], es
                 ret
strategy         endp
;
;
;        インタラプトエントリの処理
;
int_entry        proc    far
                 push    ax                ;レジスタを保存する
                 push    bx
                 push    cx
                 push    dx
                 push    si
                 push    di
                 push    bp
                 push    ds
                 push    es
                 lds     si, cs:[cmd_pack_ptr]
                 mov     bl, [si.req_code]
                 cmp     bl, 9
                 ja      cmd_err
                 xor     bh, bh
                 shl     bx, 1
                 mov     ax, offset cmd_table
                 add     bx, ax
                 mov     ax, cs
                 mov     ds, ax
                 jmp     [bx]              ;コマンドに応じて分岐する
;
cmd_err:         mov     ax, 8103h         ;ありえないコマンド
                 jmp     exit1
;
exit:            mov     ah, 01h           ;正常終了への戻り口
exit1:           lds     di, cs:[cmd_pack_ptr]
exit2:           mov     [di.req_stat], ax
                 pop     es
                 pop     ds
                 pop     bp
                 pop     di
                 pop     si
                 pop     dx
                 pop     cx
                 pop     bx
                 pop     ax
                 ret
;
;        メディアチェックは常に1（メディア変更無し）を返す
;
med_chk:         lds     bx, cs:[cmd_pack_ptr]
                 mov     [bx.mchk_val], 1
                 jmp     exit
;
;        常に単一のBPBを返すようにすれば良い。
;
get_bpb:         lds     bx, cs:[cmd_pack_ptr]
                 mov     [bx.bbpb_media_id], MEDIA_ID
                 mov     [bx.bbpb_bpbptr.fp_off], offset BPB
                 mov     [bx.bbpb_bpbptr.fp_seg], cs
                 jmp     exit
;
;        ディスク読み込み
;
dsk_read:        lds     si, cs:[cmd_pack_ptr]
                 mov     cx, [si.rw_count]
                 mov     dx, [si.rw_sec_no]
                 les     di, [si.rw_tfr_addr]
                 mov     ax, dx
                 add     ax, cx
                 cmp     ax, SEC_MAX
                 ja      dsk_err           ;転送範囲が適切か確認
;
;        転送元アドレスと転送ワード数の計算
;        SECSIZを変更した場合は、同時に訂正の必要あり
;
                 mov     ch, cl            ;rw_count*SECSIZ/2
                 xor     cl, cl
                 mov     dh, dl            ;rw_sec_no*SECSIZ + (offset disk)
                 sal     dh, 1
                 xor     dl, dl
                 add     dx, offset disk
                 mov     si, dx
                 mov     ax, cs
                 mov     ds, ax
                 cld
                 rep     movsw             ;実際の転送
                 jmp     exit
;
;        ディスク書き込み
;
dsk_write:       lds     si, cs:[cmd_pack_ptr]
                 mov     cx, [si.rw_count]
                 mov     dx, [si.rw_sec_no]
                 lds     si, [si.rw_tfr_addr]
                 mov     ax, dx
                 add     ax, cx
                 cmp     ax, SEC_MAX
                 ja      dsk_err
;
;        転送先アドレスと転送ワード数の計算
;        SECSIZを変更した場合は、同時に訂正の必要あり
;
                 mov     ch, cl
                 xor     cl, cl
                 mov     dh, dl
                 sal     dh, 1
                 xor     dl, dl
                 add     dx, offset disk
                 mov     di, dx
                 mov     ax, cs
                 mov     es, ax
                 cld
                 rep     movsw             ;データを転送
                 jmp     exit
;
dsk_err:         lds     di, cs:[cmd_pack_ptr]
                 mov     [di.rw_count], 0      ;処理したセクタ数は0
                 mov     ax, 8108h             ;セクタが見つからない
                 jmp     exit2
;
;
;        ここからがRAMディスクとして使われるメモリ
;
disk             db      3 dup (?)         ;ブートセクタの真似
                 db      'SAMPLE  '
BPB              dw      SEC_SIZ           ;Byte/sector
                 db      1                 ;sector/cluster
                 dw      1                 ;reserved sector
                 db      1                 ;FAT number
                 dw      64                ;root directory
                 dw      SEC_MAX           ;sector number
                 db      MEDIA_ID          ;media descripter
                 dw      1                 ;sector/FAT
                 dw      8
                 dw      1
                 dw      0
bpb1             db      SEC_SIZ - (bpb1 - disk) dup (?)
disk1            db      (SEC_MAX - 1)*SEC_SIZ dup (?)
;
```


NEC PC-98シリーズ一覧
PC-98XL2（1987/9発表）●CPU:V30(8MHz)/80386(16MHz)●メインメモリ：1.6MB●スロット数：4●備考：ハイレゾモデル，40MBHDD内蔵

```
int_entry       endp
;
;
;       デバイスドライバの初期化部分
;
init:           mov     ax, cs
                mov     ds, ax
                mov     es, ax
                mov     dx, offset init_msg     ;メッセージ出力
                mov     ah, 09h
                int     21h
;
                xor     ax, ax                  ;RAMディスクをフォーマット
                mov     di, offset disk1        ;FATとRootディレクトリを初期化
                mov     cx, (SEC_SIZ*3)/2
                cld
                rep     stosw
                mov     al, 0ffh
                mov     di, offset disk1
                stosb
                stosb
                stosb
;
                lds     bx, [cmd_pack_ptr]              ;MS-DOSへ返すべき値をセット
                mov     [bx.init_unit], 1               ;サポートしているユニット数1
                mov     [bx.init_end_addr.fp_off], offset init
                mov     [bx.init_end_addr.fp_seg], cs
                mov     [bx.init_bpb_addr.fp_off], offset bpb_tbl
                mov     [bx.init_bpb_addr.fp_seg], cs
                jmp     exit
;
init_msg        db      0dh, 0ah, "Sample RAM disk driver.", 0dh, 0ah, "$"
;
code            ends
                end
```

しれない．ここでは，BPB自身を先頭セクタに埋め込んでしまったが，きちんとしたものを作るときにはBPBを先頭セクタにコピーする必要がある．

デバイスドライバのデバッグ

デバイスドライバをデバッグするのは，手間のかかる作業だ．そこで，最後にデバッグをできるだけ楽にする工夫をいくつか紹介しよう．

●デバッグの工夫

デバイスドライバのデバッグはめんどうだが，やらずに済ますわけにはいかない．そこで，デバイスドライバとしてデバッグしなければならないところをできるだけ減らすことが重要だ．

デバイスドライバは，DOSとのインターフェース部分と，デバイスの操作部分とに大きくわけられる．DOSとのインターフェース部分というのは，デバイスドライバの形式でないとデバッグできないが，デバイスの操作部分，実際の仕事をする部分というのは，サブルーチンのかたちに独立させて，一般のプログラムの形式で実行させることが可能である．したがって，その段階でバグ取りをすればよろしい．万が一，デバイスドライバとして使っている間に変な挙動を見つけてしまっても，実際の仕事をしている部分のバグなら，前にもどって症状を確認してデバッグすることができる．

とくに，ハードウェアも自分で開発したような場合，ハードのほうのテスト用プログラムなどを作ってあるはずだから，それを昇格させればよい．また，DOSとのインターフェース部分は，デバイスドライバ特有の部分だが，ドライバの種類によって内容が変わることはほとんどない．ここにあげたサンプルなどをそっくり真似てしまえば問題は少なくてすむ．

デバイスドライバは，一般にBIN形式で書くことになっている（MS-DOSバージョン3.0以降はEXE形式も可能である）．ふつうはTINYモデルのプログラムのつもりで書くことになるのだが，セグメントはつねに意識しなければならない．なぜかというと，MS-DOSから渡されるコマンドパケットのアドレスは遠い遠い場所にあるかもしれないからだ．独立したプログラムとしてデバッグが終わったものが，デバイスドライバになると動作しなかったら，まずはセグメントのミスがないか調べたほうがよい．

また，デバッグ中は起動用のシステムの入ったフロッピーディスクをそばに用意しておこう．ふだん利用しているハードディスクのconfig.sysに，バグのあるデバイスドライバを指定してしまったら，ハードディスクから立ちあがらなくなってしまうかもしれない．もちろん，ハードディスクのconfig.sysはいじらずに，テスト用のフロッピーを用意しておくほうがベターだが，とりあえず暴走しない程度にしあがってくると，ハードディスクから立ちあげなおすのがめんどうになるものだ．

［参考文献］

(1)『MS-DOS3.3Bプログラマーズリファレンスマニュアル Vol.1』，日本電気．
(2)井芹陽一，「MS-DOSのデバイス・ドライバの役割」，『トラ技コンピュータ』，1989年11月号．
(3)藤井敦，「ブロック型デバイス・ドライバとRAMディスクの作り方」，『トラ技コンピュータ』，1989年11月号．
(4)石垣武彦，「キャラクタ型と常駐型デバイスドライバの作り方」，『トラ技コンピュータ』，1989年11月号．
(5)星野操，「PC98LT用RAMディスクの製作」，『プロセッサ』，1989年8月号．
(6)二上貴夫，「PC-286L/PC-98用ホスト・アダプタとデバイス・ドライバの製作」，『最新SCSIマニュアル』，CQ出版社．
(7)宇野沢成夫 橘屋鶴蔵 鈴木洋，「自分で作るSCSIのハード&ソフト」，『トランジスタ技術SPECIAL』，No.27．

特集
最新「デバイスドライバ」研究

秋山　登

C言語によるデバイスドライバ作成法

アセンブラが苦手でもデバイスドライバは書ける



デバイスドライバは，アセンブリ言語で記述するのが一般的である．これは，デバイスドライバがハードウェアに密着したプログラムであることを考えれば，ごく当然のことといえる．しかし，アセンブリ言語のみで記述されたプログラムの保守は，かなり困難を伴うものであるし，あまり歓迎されるものではない．そこで，現在パーソナルコンピュータ上でもっとも広く使われていると思われるC言語を用いてデバイスドライバを作成する方法を紹介する．アセンブリ言語を一部に用いてはいるものの，アルゴリズムの大半をC言語で記述しているので，アセンブリ言語に不案内な読者でもデバイスドライバの構造を理解することができるだろう．サンプルプログラムとして，ビープ音で音階を奏でるデバイスドライバを作成した．

MS-DOSのデバイスドライバは，普通アセンブリ言語で作成されます．ところが，アセンブリ言語で作成されたプログラムは，ドキュメントをきちんと残しておかないとあとで見直したり，保守をするのがたいへんです．アセンブリ言語の苦手なプログラマにとってデバイスドライバは，なかなか手の届かない存在になりかねません．

そこで，本文ではC言語によるビープ音を利用した音階発生用のデバイスドライバの作成を試みてみます．残念なことにすべてをC言語で書くということは無理がありますので，ここで紹介するデバイスドライバでは，一部アセンブリ言語を使用しています．しかし，デバイスドライバの大半のアルゴリズムはC言語で記述していますので，C言語の得意な諸氏には十分理解していただけると思います．

デバイスドライバと高級言語

デバイスドライバとアプリケーションの違い

高級言語の目的は，一般にOS上で走るアプリケーションプログラムを作成することにあります．すなわち，MS-DOSでいえばCOMファイルやEXEファイルを作成することにほかなりません．ところが，デバイスドライバはMS-DOSの一部として機能するプログラムで，アプリケーションプログラムとはかなり異なった環境で動くことになります．実行ファイルの形式もCOMファイルやEXEファイルとは異なります．MS-DOSにおける，アプリケーションプログラムとデバイスドライバの違いを具体的にまとめると次のようになります．

●実行ファイル形式の相違

デバイスドライバは，図1に示すように先頭にデバイスヘッダと呼ばれるデータのついたバイナリファイルです．高級言語からこのようなデータの位置を制御することは困難です．

●起動時のパラメータの受け渡し

デバイスドライバの中のデバイスストラテジルーチンは，リクエストヘッダへのポインタをES:BXのレジスタで受け取ります．高級言語では，レジスタを指定してパラメータを直接受け取ることのできる機能は普通備わっていません．

●動作環境

高級言語で書かれたプログラムは，専用の環境設定が必要です．環境設定は，プログラムの実行開始直後にプログラマからは見えないルーチンで行われます．たとえば，C言語の場合には環境設定を行うルーチンは普通crt0という名前がついていて，リンク時に自動的にリンクされ，このルーチンの中からmain関数が呼び出されます．

環境設定にはOSの呼び出しをともないますが，MS-DOSの場合，MS-DOSがリエントラント構造になっていないため，デバイスドライバは環境設定のルーチンを使用することができません．なぜなら，デバイスドライバはMS-DOSから呼び出されるからです．このため，デバイスドライバを高級言語で記述すると，高級言語に備わっている便利な機能はほとんど使用できなくなってしまいます．

●実行時間とタイミング

高級言語で書かれたプログラムは，アセンブリ言語で書かれたプログラムより実行に時間がかかり，サイズも大きくなる傾向があります．デバイスドライバの性格上，実行時間の増加はシステム全体の性能に影響しますし，サイズの増加はアプリケーションプログラムの動作に影響を与えますので，好ましくありません．さらに，出力されるコードをプログラマはあまり制御で

図1 デバイスドライバの実行ファイルの構造

デバイスヘッダ	
デバイスドライバリンクのためのポインタスペース	2ワード
デバイスの属性コード	1ワード
デバイスストラテジルーチンのエントリポイント	1ワード
デバイス割り込みルーチンのエントリポイント	1ワード
デバイス名あるいはユニット数	8バイト
デバイスストラテジルーチン本体 デバイス割り込みルーチン本体 作業用データ領域など	

リスト1

```
IN      AL,02H
JMP     SHORT $+2
JMP     SHORT $+2
AND     AL,0FEH
OUT     02H,AL
```

きませんので，リスト1のJMP SHORT $+2のようにハードウェアの問題から起きる微妙なタイミングの調整を行うことが困難になります．

Cでデバイスドライバを作る環境を整える
C言語とアセンブリ言語の使い分け

前述したように，高級言語でデバイスドライバを記述することには問題点も多いのですが，手軽さ，とくに可読性のよさは捨て難いものがあります．そこで，例をMS-C 6.0に絞り，キャラクタ型のデバイスドライバをC言語で作成するための環境を整えることにします．

いろいろな方法が考えられますが，ここではファイル形式を整え，C言語のための環境設定をする部分をアセンブリ言語で記述し，残りの部分をC言語で記述する方法をとります．C言語で記述した部分は，そのまま利用できるように配慮します．

ここで述べる方法以外には，C言語で作成したプログラムをアセンブリ言語のソースコードに落とし，それに手を加える方法などがあります．

アセンブリ言語の部分

アセンブリ言語の部分で行うことは次のとおりです．

セグメントの配置

MS-Cのスモールあるいはタイニーモデルで生成されるオブジェクトのセグメントは，図2のような順序になっています．

セグメントは全部で4つありますが，_DATA，CONST，および_BSSセグメントはDGROUPにひとまとめにされるので，実質的には2つのセグメントが存在していることになります．

_DATAセグメントとCONSTセグメントの違いは，書き換えられる可能性があるかないかによります．タイニーモデルでは，環境設定ルーチンの中でさらに_TEXTセグメントがDGROUPにまとめられ，スタックセグメントもDGROUPに加えられて実質的に1つのセグメントになります．そこで，このタイニーモデルのやり方にならって，アセンブリ言語の部分では，図3のようなセグメントの配置を行います．なお，スタックセグメントの名前は任意ですが，結合属性にSTACKを指定するとEXE2BINを実行する際，問題を起こしますので注意します．

デバイスヘッダ

_TEXTセグメントを最初に配置しますので，_TEXTセグメントの先頭に図1の構造のデバイスヘッダをつけます．

デバイスストラテジルーチン

C言語で書くまでもなく，非常に簡単なルーチンなのでアセンブリ言語で記述します．リクエストヘッダへのポインタは，C言語からも参照できるようPUBLICにします．

C言語の関数を呼び出すための環境設定

図2　スモール/タイニーモデルのセグメント配置

_TEXT		コード
DGROUP	_DATA	初期化されたデータ領域
	CONST	初期化されたデータ領域
	_BSS	初期化されないデータ領域

図3　デバイスドライバのセグメント配置

DGROUP	_TEXT	デバイスヘッダと一部のデータおよびコード
	_DATA	初期化されたデータ(書き換え可能)
	CONST	初期化されたデータ(書き換え不可能)
	_BSS	初期化されないデータ
	_STACK	スタック

C言語の環境設定ルーチンで行われる最低限の環境設定を行います．MS-C 6.0では，次の項目について環境設定を行う必要があります．

①スタックの設定（割り込み処理では不要）

②DSレジスタの値をDGROUPの先頭のパラグラフアドレスに設定

③ディレクションフラグをクリアする

④レジスタの待避と復旧を行う

●●● 最終アドレス

デバイスドライバに対する初期化コマンドで，デバイスドライバ自身の最終アドレスをMS-DOSに返す必要があります．このため，スタックセグメントの最終番地をDGROUP先頭からのポインタとして，C言語の関数から参照できるようにします．

●●● 割り込みハンドラ

デバイスドライバでは，ハードウェア割り込みを使用する場合がよくあります．MS-Cでは，_interruptを使用することによって割り込みハンドラを作成することができますが，この中でリスト2のようなコードを生成します．

このコードは，セグメントアドレスをイミーディエットで操作しています（SEG DGROUPの部分）が，EXE2BINを使用してバイナリファイルを生成するときに問題を生じます．そこで，割り込みハンドラの環境設定の部分はアセンブリ言語で記述し，C言語の通常の関数で割り込みハンドラの本体を作成するようにします．C言語から割り込みハンドラの本体を登録するための領域を確保し，C言語の関数が割り込みベクトルを設定するために，割り込みハンドラのアセンブリ言語部分の先頭のポインタを参照できるよう

リスト 2

```
        MOV     AX,SEG DGROUP
        MOV     DS,AX
```

リスト 3　デバイスドライバのアセンブリ言語部分(CHRDRV0.ASM)

```
        PAGE    69,132
        NAME    CHRDRV0

        PUBLIC  DVHDTOP         ; リンク確認用
        PUBLIC  _rqhdp          ; リクエストヘッダへのポインタ
        PUBLIC  _lstadr         ; デバイスドライバの最終アドレス
        PUBLIC  _inthnd         ; 割込ハンドラのエントリ
        PUBLIC  _intfnc         ; Cによる割込ハンドラへのポインタ
        PUBLIC  _msgout         ; 組み込みメッセージ表示
        PUBLIC  STKTOP          ; リンク確認用
        EXTRN   _drvmain:NEAR
;
; セグメントの配置の宣言
;
_TEXT   SEGMENT WORD PUBLIC 'CODE'
_TEXT   ENDS
_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA   ENDS
CONST   SEGMENT WORD PUBLIC 'CONST'
CONST   ENDS
_BSS    SEGMENT WORD PUBLIC 'BSS'
_BSS    ENDS
_STACK  SEGMENT PARA PUBLIC 'STACK'
_STACK  ENDS
;
; すべてのセグメントを１つにまとめる
;
DGROUP  GROUP   _TEXT,_DATA,CONST,_BSS,_STACK
;
_TEXT   SEGMENT         ; デバイスドライバの先頭に配置
        ASSUME  CS:DGROUP,DS:DGROUP
;
; キャラクタデバイス用デバイスヘッダ
;
DVHDTOP:
        DD      -1
        DW      1100100000000000B       ; IOCTL,OPEN/CLOSEをサポート
        DW      OFFSET DGROUP:DRVSTR    ; デバイスストラテジエントリ
        DW      OFFSET DGROUP:DRVINT    ; デバイス割込エントリ
        DB      '$TONEDRV'              ; デバイス名
;
_rqhdp  DW      0       ; リクエストヘッダのアドレス
        DW      0
_lstadr DW      0       ; デバイスドライバの最終アドレス
        DW      0
SPSAV   DW      0       ; デバイス割込時のスタック待避
SSSAV   DW      0
_inthnd DW      0       ; 割込ハンドラのアドレス
        DW      0
_intfnc DW      0       ; C言語割込ハンドラのアドレス
;
; デバイスストラテジルーチン
;
DRVSTR  PROC    FAR
;
        MOV     CS:_rqhdp,BX
        MOV     CS:_rqhdp+2,ES
        RET
;
DRVSTR  ENDP
;
; デバイス割込ルーチン
;
DRVINT  PROC    FAR
;
        PUSH    AX              ; レジスタの待避(1)
        PUSH    BX
        MOV     CS:SPSAV,SP     ; スタックの待避
        MOV     CS:SSSAV,SS
        PUSHF                   ; 割込禁止(1)
        POP     BX
        CLI
        MOV     AX,CS           ; C関数用スタックの設定
        MOV     SS,AX
        MOV     SP,OFFSET DGROUP:STKTOP
        PUSH    BX              ; 割込状態の復元(1)
        POPF
```

```
        MOV     BX,SP           ; デバイスドライバの最終アドレス
        DEC     BX
        MOV     CS:_lstadr,BX
        MOV     CS:_lstadr+2,AX
        PUSH    CX              ; レジスタの待避(2)
        PUSH    DX
        PUSH    SI
        PUSH    DI
        PUSH    BP
        PUSH    DS
        PUSH    ES
        MOV     DS,AX           ; C関数用DSレジスタの設定(1)
        MOV     BX,OFFSET DGROUP:INTHND ; 割込ハンドラのアドレス
        MOV     _inthnd,BX
        MOV     _inthnd+2,AX
        CLD                     ; C関数用ディレクションフラッグの設定(1)
        CALL    _drvmain        ; Cによるデバイス割込ルーチンの呼出
        POP     ES              ; レジスタの復旧(2)
        POP     DS
        POP     BP
        POP     DI
        POP     SI
        POP     DX
        POP     CX
        PUSHF                   ; 割込禁止(2)
        POP     BX
        CLI
        MOV     SS,CS:SSSAV     ; スタックポインタの復旧
        MOV     SP,CS:SPSAV
        PUSH    BX              ; 割込状態の復元(2)
        POPF
        POP     BX              ; レジスタの復旧(1)
        POP     AX
        RET
;
DRVINT  ENDP
;
; 割込ハンドラ
;
INTHND  PROC    FAR
;
        PUSH    AX              ; レジスタの待避(3)
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        PUSH    DI
        PUSH    BP
        PUSH    DS
        PUSH    ES
        MOV     AX,CS           ; C関数用DSレジスタの設定(2)
        MOV     DS,AX
        CLD                     ; C関数用ディレクションフラッグの設定(2)
        CALL    _intfnc         ; Cによる割込ハンドラの呼出
        POP     ES              ; レジスタの復旧(3)
        POP     DS
        POP     BP
        POP     DI
        POP     SI
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        IRET
;
INTHND  ENDP
;
; メッセージの表示
;
_msgout PROC    NEAR
;
        PUSH    BP
        MOV     BP,SP
        MOV     DX,4[BP]        ; byte _near *
        MOV     AH,09H
        INT     21H
        POP     BP
        RET
;
_msgout ENDP
;
_TEXT   ENDS
;
_STACK  SEGMENT         ; デバイスドライバの最後に配置
;
        DW      256 DUP(0)
STKTOP  EQU     $
;
_STACK  ENDS
;
        END     DVHDTOP ; メインモジュールにするためのダミー
```

にしておかなければなりません．

ここで述べたようなコードは，このほかの場所でも生成されることがあります．その場合には，共用体を使用してオフセット部分とセグメント部分を合成して far ポインタを作成するような工夫が必要になります．

以上のような項目を実現したものが，リスト 3 の CHRDRV0.ASM です．C言語のコンパイルはタイニーモデルを使用することを前提にしています．デバイス割り込みルーチンの本体の関数名は drvmain です．C言語でグローバルシンボルを記述する際は，アンダースコア（_）が不要ですので注意してください．

リスト 3 は，キャラクタデバイス用で，デバイス名は$TONEDRV です．デバイスの属性，デバイス名，グローバルシンボルなどは必要に応じて変更してかまいません．_STACK セグメントのサイズは，C言語の関数のスタックの使用量に応じて変更する必要があります．

C言語の部分

アセンブラの部分で，ほぼC言語の部分のコーディングスタイルが決定しました．C言語で記述するデバイス割り込みルーチンの本体は drvmain です．ハードウェア割り込みを使用する場合には，intfnc にその関数への near ポインタを設定します．

アセンブリ言語の部分とインターフェースをとるためのヘッダファイル CHRDRV0.H をリスト 4 に示します．この中で，構造体 rqhdr はキャラクタデバイスに必要なリクエストヘッダの内容へアクセスするためのものです．構造体の中には，無名の共用体が

NEC PC-98 シリーズ一覧
PC-9801LV21（1988/2発表）●CPU:V30(8/10MHz)●メインメモリ：640KB●備考：ラップトップ

リスト 4　アセンブリ言語部分を利用するためのヘッダ(CHRDRV0.H)

```
typedef unsigned char byte;
typedef unsigned int  word;
typedef unsigned long dword;

struct rqhdr {
        byte    rqleng,           /* リクエストヘッダの長さ */
                rqucod,           /* ユニットコード */
                rqcomd;           /* コマンドコード */
        word    rqstts;           /* ステータス */
        void    _far *rqdosq,     /* 予約領域 */
                _far *rqdevq;     /* 予約領域 */
        byte    rqprm0;           /* ユニット数・メディアディスクリプタ */
        void    _far *rqprm1;     /* エンドアドレス・転送アドレス */
        union {
                void    _far *rqprm20;  /* BPB配列へのポインタ */
                word    rqprm21[2];     /* [0]=データカウント */
        };
};

extern struct rqhdr _far *rqhdp;/* リクエストヘッダへのポインタ */
extern void _far *lstadr;         /* デバイスドライバの最終アドレス格納場所 */
extern void (*intfnc)();          /* Cによる割込ハンドラ登録場所 */
extern void (_far *inthnd)();     /* 実際の割込エントリ格納場所 */

void drvmain(void);               /* Cによるデバイス割込ルーチン */
void msgout(byte *msg);           /* 文字列表示関数 */

```

含まれていますのでほかのCに移植する際は注意してください．また，この構造体を利用するためには，メンバをパックしてコンパイルしなければなりません．キーワード_farはfarポインタを宣言するのに使用しています．

アセンブリ言語の部分で行った環境設定が必要最小限であったためと，MS-DOSの呼び出しが初期化のときのインストールメッセージの表示に限られるため，C言語の便利なライブラリ関数はほとんど使用できなくなります．使用できないライブラリ関数の目安は次のようなものです．

①入出力を伴う関数（errnoにエラーコードを設定する関数），printf，getc，open，fopenなど

②ヒープを操作したり使用する関数，malloc，strdup，exec，spawnなど

③本来の環境設定ルーチンに含まれる変数を参照する関数，intdos，int86など

使用できるライブラリ関数を調べるには，関数のレファレンスマニュアルを参照するか，それでも不明ならば実際に使用してリンク時のエラーメッセージで調べるか，あるいはSYMDEBのようなデバッガで実際のコードを確認するかのいずれかです．現実には，ライブラリ関数の使用を極力避けるのが賢明です．

ビープ音発生のデバイスドライバを作る
単音だがメロディ演奏も可能

環境が整ったところで，具体的なデバイスドライバの作成にチャレンジしてみましょう．作成するデバイスドライバは，PC-9801とPC-9801E，F，Mを除くPC-9801シリーズの機種で，ビープ音発生用のハードウェアを操作して音楽の音階を一定時間発生させるものです．単音ですが，発生時間，休止，音階を操作して簡単なメロディ演奏ができるものです．もちろんモールス符号の発生もできます．

使用するハードウェアの構成を抜き出すと，図4のようになります．中心はカウンタタイマのLSI8253Aです．このLSIのチャンネル1をモード3で使用して，このチャンネルのカウント数を変更することによりスピーカに送る信号の周波数を変化させます．

チャンネル0はモード2で使用して，マスタ側のPIC8259Aを経由して10ミリ秒ごとに割り込みを発生させ，音の長さを制御します．スピーカのON/OFFはシステムポートとして使用される8255AのCポートビット3で行います．

デバイスドライバに出力するデータは，6バイト固定です．6バイト出力されると，残りのバイトは無視して処理を開始します．最初の4バイトは音階データへのfarポインタ，最後の2バイトは音階データのバイト数を指定するパラメータです．いずれも下位バイトから送ります．このデータのようすを図5に示します．

6バイトの出力データを受け取ると，デバイスドライバは呼び出し元へ復帰して，以降は8253Aからの10ミリ秒ごとの割り込みによって処理を続けます．このときに処理される音階のデータはunsigned char型で表1のとおりです．この表にない0x3Cから0xFEまでのデータは休符の働きをします．0xFFは，音の持続時間を次のバイトの数値により10ミリ秒単位で設定します．たとえば，0xFF，0x0Aと音階データが続いた場合，音の持続時間は100ミリ秒に設定されます．

このデバイスドライバは突き放し処理を行うため，処理の終了を知る手段が必要です．デバイスドライバは，つねに1バイトのデータが入力可能で，

図 4
トーン発生ハードウェアの構成図

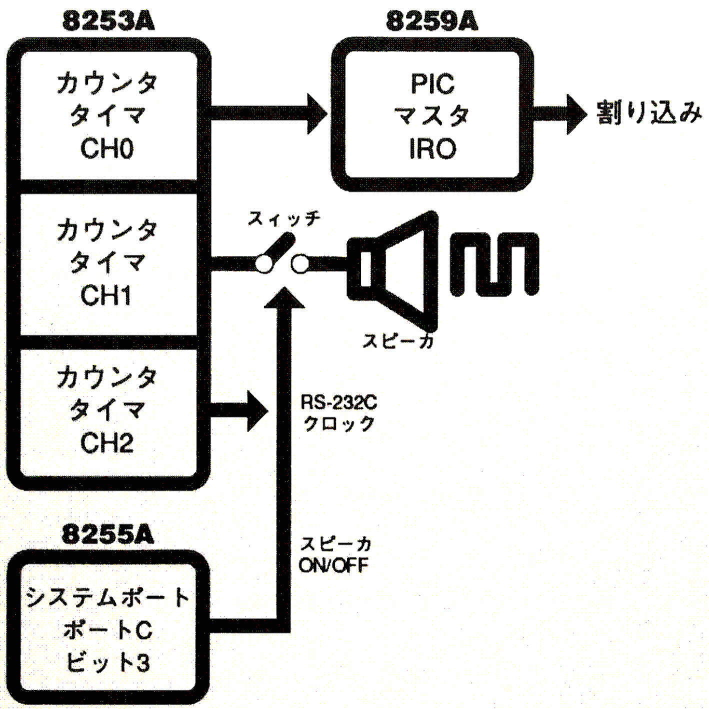


図 5
デバイスドライバへの出力データ

先頭 最終

音階データへのポインタ		音階データのバイト数	
オフセット	セグメント	下位バイト	上位バイト

入力されたデータが0x00のときは処理が行われていないことを，0x01のときは処理中であることを示します．

プログラム3は，このデバイスドライバをMS-C 6.0用にコーディングしたTONE.Cで，9個の関数から構成されています．各関数の機能を簡単に説明しておきます．

init

デバイスドライバに対するコマンドコード0の処理，すなわちデバイスドライバの初期化を行います．変数の初期化，レゾリューションの判定，クロック系統の判定，組み込みメッセージの表示を行います．クロック系統の判定は不完全なので，組み込みメッセージに使用している機種と異なるクロック系統が表示された場合には，クロックの判定部分を8MHz系か10MHz系かのいずれかに固定してください．この関数は組み込みのとき一度だけしか呼ばれないため切り放しが可能ですが，このデバイスドライバでは切り放しは行っていません．

drvmain

アセンブリ言語の部分からデバイス割り込みルーチンの本体として呼び出されます．この関数は，デバイスドライバへのコマンドの切り分けと大半の

表 1 音階データ

音階	音階データ				
A	0x00(165Hz)	0x0C(330Hz)	0x18(660Hz)	0x24(1320Hz)	0x30(2640Hz)
A#	0x01	0x0D	0x19	0x25	0x31
B	0x02	0x0E	0x1A	0x26	0x32
C	0x03	0x0F	0x1B	0x27	0x33
C#	0x04	0x10	0x1C	0x28	0x34
D	0x05	0x11	0x1D	0x29	0x35
D#	0x06	0x12	0x1E	0x2A	0x36
E	0x07	0x13	0x1F	0x2B	0x37
F	0x08	0x14	0x20	0x2C	0x38
F#	0x09	0x15	0x21	0x2D	0x39
G	0x0A	0x16	0x22	0x2E	0x3A
G#	0x0B	0x17	0x23	0x2F	0x3B

コマンドの処理を行います．

●●● timerint

タイマのチャンネル0からの10ミリ秒おきの割り込みを処理します．timcntを割り込みのたびにカウントダウンして，0になると次のデータをタイマのチャンネル1にセットします．同時に，timcnsに音階データから設定されている値をtimcntに再設定します．音階データがなくなると，timerstp関数を呼び出して処理を終了します．

●●● timerstt

出力データを6バイト受け取ると，output関数から呼び出されて割り込み環境を整えた後，タイマをスタートさせて音階データの処理を開始します．

●●● timerstp

割り込み環境を元に戻し，スピーカをOFFにしてタイマを停止させます．

●●● nextdata

次の音階データをタイマのカウント値に変換して共用体bwにセットします．音階データの中に音の持続時間のデータがあるとtimcnsに設定します．音階データがある間は－1を返し，音階データがなくなると0を返します．

●●● iocsts

デバイスドライバに対するコマンドコード3と4，すなわち入力データをMS-DOSへ引き渡します．timenaを1バイトの入力データとしてMS-DOSへ返します．timenaは，音階データを処理しているときは1で，処理していないときは0です．

●●● output

デバイスドライバに対するコマンド

リスト5　音階発生のデバイスドライバ(TONE.C)

```
#include <CONIO.H>
#include <DOS.H>
#include "CHRDRV0.H"

#define MAXTONE 60                     /* 音階データの最大数 */
#define VECT08 0x00000020              /* 割込ベクトル08Hのアドレス */
#define RESOL  0x00000500              /* レゾリューション識別のアドレス */
#define PICEOI 0x0000                  /* PICマスタにEOIを送るアドレス */
#define PICIMR 0x0002                  /* PICマスタのIMRアドレス */
#define TIMER0 0x0071                  /* タイマCH0のカウント値設定アドレス */
#define TIMER1H 0x0073                 /* タイマCH1のカウント値設定アドレス */
#define TIMER1L 0x3FDB                 /* タイマCH1のカウント値設定アドレス */
#define TIMERMH 0x0077                 /* タイマモード設定アドレス */
#define TIMERML 0x3FDF                 /* タイマモード設定アドレス */
#define SPSW   0x0037                  /* スピーカON/OFFのポートアドレス */
#define SPON   0x06                    /* スピーカONのデータ */
#define SPOFF  0x07                    /* スピーカOFFのデータ */

static word                  /* 音階のデータ，ttbl0:8MHz系，ttbl1:10MHz系 */
        ttbl0[MAXTONE] = { 12102,11423,10782,10176, 9605, 9066,
                            8557, 8077, 7624, 7196, 6792, 6411,
                            6051, 5711, 5391, 5088, 4803, 4533,
                            4279, 4039, 3812, 3598, 3396, 3205,
                            3025, 2856, 2695, 2544, 2401, 2267,
                            2139, 2019, 1906, 1799, 1698, 1603,
                            1513, 1428, 1348, 1272, 1201, 1133,
                            1070, 1010,  953,  899,  849,  801,
                             756,  714,  674,  636,  600,  567,
                             535,  505,  476,  450,  424,  401 },
        ttbl1[MAXTONE] = { 14895,14059,13270,12525,11822,11158,
                           10532, 9941, 9383, 8856, 8359, 7890,
                            7447, 7029, 6635, 6262, 5911, 5579,
                            5266, 4970, 4691, 4428, 4180, 3945,
                            3724, 3515, 3317, 3131, 2955, 2790,
                            2633, 2485, 2346, 2214, 2090, 1973,
                            1862, 1757, 1659, 1566, 1478, 1395,
                            1317, 1243, 1173, 1107, 1045,  986,
                             931,  879,  829,  783,  739,  697,
                             658,  621,  586,  554,  522,  493 };
static byte
        _far *outpnt,   /* 出力データポインタ */
        _far *ioppnt;   /* 入出力ポインタ */
static word
        _far *datpnt;   /* データへのポインタ */
static word
        *tblpnt,        /* 音階テーブル選択 */
        opncnt,         /* オープン・クローズのカウント */
        outcnt,         /* 出力データカウンタ */
        datcnt,         /* データの残カウント */
        spflag,         /* 0=スピーカOFF，1=スピーカON */
        rslflg,         /* 0=ノーマル，1=ハイレゾリューション */
        timmod,         /* タイマ#1モード設定ポートアドレス */
        timdat,         /* タイマ#1カウントデータポートアドレス */
        timena,         /* 0=タイマ停止，1=タイマ動作 */
        timcns,         /* トーンの持続時間 */
        timval,         /* 10msのカウンタ値 */
        timold,         /* 2kHzのカウンタ値 */
        timcnt,         /* 持続時間測定カウンタ */
        imrsav;         /* PIC IMRの待避復元 */
static union {          /* ワード，バイト分解合成 */
        byte    bx[2];
        word    wx;
} bw;
static union {          /* ポインタをバイトデータより合成 */
        byte    bx[4];
        byte    _far *px;
} bd;

static void
        (_far *sav08)(),        /* 割込ベクトル08Hの待避復元 */
        (_far * _far *vct08)(); /* 割込ベクトル08Hへのポインタ */

static void init(void);
static void timerint(void);
static void timerstt(void);
static void timerstp(void);
static int nextdata(void);
static void iocsts(void);
static void output(void);
static void somewait(void);

/* デバイス割込ルーチン */
void drvmain(void) {

        word    cmdcd;

/* コマンドの切り分けと処理 */
        cmdcd=(word)rqhdp->rqcomd;
        switch(cmdcd) {
        case  0:                        /* INIT */
                init();
                break;
        case  3:                        /* IOCTL INPUT */
        case  4:                        /* INPUT */
                iocsts();
                break;
        case  5:                        /* NON-DESTRUCTIVE INPUT NO WAIT */
                rqhdp->rqprm0=(byte)timena;     /* 入力データ */
        case  6:                        /* INPUT STATUS */
        case  7:                        /* INPUT FLUSH */
                rqhdp->rqstts=0x0100;           /* ステータス=DONE */
                break;
        case  8:                        /* OUTPUT */
        case  9:                        /* OUTPUT WITH VERIFY */
                output();
```

コード8と9，すなわち出力データの受け取りを処理します．最初の4バイトを音階データへのポインタとしてoutpntに，次の2バイトを音階データのバイト数としてdatcntにセットします．そのあとtimersttを呼び出して音階データの処理を開始させます．一度に6バイト以上の出力データが送られてきたときには，残りのデータは無視します．

●●● somewait

入出力命令間の時間をあけるために呼び出します．80286CPUの場合，20クロックサイクル以上の間隔があきます．速度の速い機種で間隔がたりない場合は，この関数を呼ぶ回数を増やしてください．

アセンブリ言語の部分(CHRDRV0.ASM)とC言語の部分(TONE.C)を結合して，デバイスドライバに仕上げるためには，リスト6のバッチプログラムを走らせます．コンパイル時に，構造体のメンバをパックするためのオプション/Zpを忘れないよう注意します．スタックチェックは使用できませんので/Gsも指定する必要があります．

リスト6のバッチプログラムで，TONE.SYSというデバイスドライバの実行ファイルができあがります．あとはCONFIG.SYSに，DEVICE = TONE.SYSを追加すれば使用できます．リスト7は，このデバイスドライバのテストプログラムです．ドレミファソラシドの音階を発生させます．

最後にデバッグについて簡単に触れておきます．もっとも簡単なのは，SYMDEBを使用する方法です．

デバイスドライバをSYMDEBにロードして，ストラテジルーチンと割り込みルーチンのエントリポイントを

```
			break;
		case 10:			/* OUTPUT STATUS */
			if(timena)
				rqhdp->rqstts=0x0300;	/* ステータス=BUSY|DONE */
			else
				rqhdp->rqstts=0x0100;	/* ステータス=DONE */
			break;
		case 11:			/* OUTPUT FLUSH */
			if(timena)
				timerstp();
			outcnt=0;
			rqhdp->rqstts=0x0100;		/* ステータス=DONE */
			break;
		case 13:			/* DEVICE OPEN */
			opncnt++;
			rqhdp->rqstts=0x0100;		/* ステータス=DONE */
			break;
		case 14:			/* DEVICE CLOSE */
			if((--opncnt)==0) {
				if(timena)
					timerstp();
			}
			outcnt=0;
			rqhdp->rqstts=0x0100;		/* ステータス=DONE */
			break;
		case  1:			/* MEDIA CHECK */
		case  2:			/* BUILD BPB */
		case 12:			/* IOCTL OUTPUT */
		case 15:			/* REMOVABLE MEDIA */
		case 16:			/* OUTPUT UNTIL BUSY */
		default:
			rqhdp->rqstts=0x8103;		/* ステータス=ERROR| */
			break;				/* DONE|不正コマンド */
		}
}

/* 初期化 */
static void init(void) {

		int	wk;

/* 変数の初期化 */
		vct08=(void(_far * _far *)())VECT08;
		intfnc=timerint;
		outcnt=0;
		datcnt=0;
		spflag=0;
		timena=0;
/* レゾリューションとクロックの判定 */
		datpnt=(word _far *)RESOL;
		if(*datpnt & 0x0800) {	/* ハイレゾリューション */
			rslflg=1;
			timmod=TIMERMH;
			timdat=TIMER1H;
			wk=inp(0x0063);
			if((wk & 0x007E)==0x007E) {
				if(wk & 0x0001) {
					timval=19968;	/* 8MHz系 */
					timold=998;
					tblpnt=ttbl0;
				}
				else {
					timval=24576;	/* 10MHz系 */
					timold=1229;
					tblpnt=ttbl1;
				}
			}
			else {
				timval=19968;		/* 8MHz系 */
				timold=998;
				tblpnt=ttbl0;
			}
		}
		else {				/* ノーマルレゾリューション */
			rslflg=0;
			timmod=TIMERML;
			timdat=TIMER1L;
			wk=inp(0x0042);
			if((wk & 0x00C0)==0x00C0) {
				timval=19968;		/* 8MHz系 */
				timold=998;
				tblpnt=ttbl0;
			}
			else if(wk & 0x0020) {
				timval=19968;		/* 8MHz系 */
				timold=998;
				tblpnt=ttbl0;
			}
			else {
				timval=24576;		/* 10MHz系 */
				timold=1229;
				tblpnt=ttbl1;
			}
		}
/* インストールメッセージの表示 */
		msgout("¥r¥nTONE DRIVER INSTALLED. ($");
		if(rslflg)
			msgout("HIGH RESOLUTION,$");
		else
			msgout("NORMAL RESOLUTION,$");
		if(timold==998)
			msgout("8/16/33MHz)¥r¥n$");
		else
			msgout("5/10/20MHz)¥r¥n$");
/* ステータス等をMS-DOSへ返す */
```

```
	rqhdp->rqprm0=1;		/* ユニット数 */
	rqhdp->rqprm1=lstadr;	/* デバイスドライバの最終アドレス */
	rqhdp->rqstts=0x0100;	/* ステータス=DONE */
}

/* タイマ割込処理 */
static void timerint(void) {

	outp(PICEOI, 0x0B);
	somewait();
	if(inp(PICEOI) & 0x0001) {			/* PICによる割込 */
		outp(PICEOI, 0x60);			/* 特定EOI */
		somewait();
		if((--timcnt)==0) {
			if(nextdata()) {
				timcnt=timcns;
				if(bw.wx==0) {	/* 休止 */
					outp(SPSW, SPOFF);
					spflag=0;
				}
				else {		/* 次のトーンデータ */
					outp(timmod, 0x76);
					somewait();
					outp(timdat, (int)bw.bx[0]);
					somewait();
					outp(timdat, (int)bw.bx[1]);
					somewait();
					if(spflag==0) {
						outp(SPSW, SPON);
						spflag=1;
					}
				}
			}
			else			/* 全データ終了 */
				timerstp();
		}
	}
}

/* タイマLSIの初期化と割込環境設定 */
static void timerstt(void) {

	timcns=10;
	if(nextdata()==0) {	/* データなし */
		timena=0;
		return;
	}
	if(bw.wx==0) {		/* 休止 */
		outp(SPSW, SPOFF);
		spflag=0;
	}
	else {			/* トーンデータをタイマCH1に設定 */
		outp(timmod, 0x76);
		somewait();
		outp(timdat, (int)bw.bx[0]);
		somewait();
		outp(timdat, (int)bw.bx[1]);
		somewait();
		outp(SPSW, SPON);
		spflag=1;
	}
	timcnt=timcns;
	sav08=*vct08;		/* 割込ベクトル設定 */
	_disable();
	*vct08=inthnd;
	imrsav=inp(PICIMR);	/* PICのIMR待避と設定 */
	somewait();
	outp(PICIMR, imrsav & 0x00FE);
	somewait();
	bw.wx=timval;
	timena=1;
	outp(TIMERMH, 0x34);	/* タイマCH0の設定(周期10msec) */
	somewait();
	outp(TIMER0, (int)bw.bx[0]);
	somewait();
	outp(TIMER0, (int)bw.bx[1]);
	somewait();
	_enable();
}

/* タイマと割込環境の後始末 */
static void timerstp(void) {

	outp(SPSW, SPOFF);			/* スピーカOFF */
	spflag=0;
	_disable();
	outp(PICIMR, imrsav);			/* PICのIMRを復旧 */
	*vct08=sav08;				/* 割込ベクトルを復旧 */
	timena=0;
	outp(TIMERMH, 0x34);			/* タイマCH0カウント停止 */
	somewait();
	bw.wx=timold;
	outp(timmod, 0x76);			/* タイマCH1を通常の値に設定 */
	somewait();
	outp(timdat, (int)bw.bx[0]);
	somewait();
	outp(timdat, (int)bw.bx[1]);
	_enable();
}

/* 次のデータを得る */
static int nextdata(void) {

	word	wk;

	while(1) {
		if(datcnt==0)
			return 0;				/* データなし */
		wk=(word)*outpnt++;
		datcnt--;
		if(wk==0x00FF) {				/* 速度データ */
			if(datcnt) {
				timcns=(word)*outpnt++;
				datcnt--;
			}
			else
				return 0;			/* データなし */
		}
		else {
			if(wk>=MAXTONE)
				bw.wx=0;			/* 休止データ */
			else
				bw.wx=*(tblpnt+wk);	/* 音階データ */
			break;
		}
	}
	return -1;						/* データあり */
}

/* 入力データ(処理状況)を返す */
static void iocsts(void) {

	ioppnt=(byte _far *)rqhdp->rqprm1;	/* 転送アドレス */
	*ioppnt=(byte)timena;			/* 入力データ=処理状況 */
	rqhdp->rqprm21[0]=1;			/* 入力バイト数=1 */
	rqhdp->rqstts=0x0100;			/* ステータス=DONE */
}

/* 出力データを受け取る */
static void output(void) {

	word	cnt;

	cnt=rqhdp->rqprm21[0];			/* 出力データの数 */
	ioppnt=rqhdp->rqprm1;			/* 出力データの位置 */
/* 出力データを6バイト受け取り,残りは読み捨てる */
	while(cnt) {
		if(outcnt<4) {
			bd.bx[outcnt++]=*ioppnt++;
			cnt--;
		}
		else if(outcnt>5)
			break;
		else {
			bw.bx[outcnt-4]=*ioppnt++;
			outcnt++;
			cnt--;
		}
	}
	if(outcnt>5) {					/* データ受け取り完了 */
		outpnt=bd.px;
		datcnt=bw.wx;
		timerstt();
		outcnt=0;
	}
	rqhdp->rqstts=0x0100;			/* ステータス=DONE */
}

/* 入出力命令間に多少の待ち時間を作る */
static void somewait(void) {
}
```

確認します．それから，空いているメモリスペースに，リクエストヘッダを作成して必要なパラメータを設定し，ES:BXに作成したリクエストヘッダへのポインタをセットして，ストラテジのエントリポイントにファーコールを行います．そして次に，割り込みルーチンのエントリポイントにファーコールを行います．すなわち，MS-DOSのかわりに自分でデバイスドライバを呼び出すわけです．このような方法では，ブレークポイントの設定やメモリダンプが自由にできます．

このほかに，デバイスドライバからVRAMに直接書き込む方法で情報を表示させることもできます．いずれにしても，デバイスドライバのデバッグはアプリケーションプログラムに比べると手間がかかりますから，最初から細心の注意を払って設計するよう心がけるべきでしょう．

デバイスドライバを高級言語で作成する方法はいろいろ考えられますが，その中からC言語による一方法を紹介しました．読者の参考になれば幸いです．

リスト 6　バッチプログラム(MAKETONE.BAT)

```
CL /AT /c /Osler /Zp /Gs /W3 /J TONE.C > TONE.ERR
IF ERRORLEVEL 1 GOTO ERR
TYPE TONE.ERR
MASM CHRDRV0,CHRDRV0,NUL/Z/B63/R/MX; > TONE.ERR
IF ERRORLEVEL 1 GOTO ERR
TYPE TONE.ERR
LINK CHRDRV0+TONE,TONE.EXE,TONE.MAP/MAP/NOI; > TONE.ERR
IF ERRORLEVEL 1 GOTO ERR
TYPE TONE.ERR
EXE2BIN TONE.EXE TONE.SYS > TONE.ERR
:ERR
TYPE TONE.ERR
```

リスト 7　デバイスドライバのテストプログラム(TESTTONE.C)

```
#include <STDLIB.H>
#include <STDIO.H>
#include <FCNTL.H>
#include <SYS¥TYPES.H>
#include <SYS¥STAT.H>
#include <IO.H>

static char
        dbuf[10];
static char
        tdat[19] = { 255, 50, 27, 29, 31,
                      32, 34, 36, 38, 39,
                     254, 39, 38, 36, 34,
                      32, 31, 29, 27 };
static union {
        char    cx[4];
        char    _far *px;
} cp;
static union {
        char    cx[2];
        unsigned int
                wx;
} cw;

void main(void) {

        int     hand,
                len;

        cp.px=(char _far *)&tdat[0];
        cw.wx=19;
        dbuf[0]=cp.cx[0];
        dbuf[1]=cp.cx[1];
        dbuf[2]=cp.cx[2];
        dbuf[3]=cp.cx[3];
        dbuf[4]=cw.cx[0];
        dbuf[5]=cw.cx[1];
        hand=open("$TONEDRV",O_RDWR|O_BINARY);
        if(hand==-1) {
                printf("$TONEDRVがオープンできません。¥n");
                exit(1);
        }
        len=write(hand,dbuf,6);
        if(len!=6) {
                printf("$TONEDRVに書き込めません。¥n");
                exit(1);
        }
        while(1) {
                len=read(hand,dbuf,1);
                if(len!=1) {
                        printf("$TONEDRVから読み込めません。¥n");
                        exit(1);
                }
                if((int)dbuf[0]==0)
                        break;
        }
        close(hand);
}

```

特集
最新「デバイスドライバ」研究



Windowsのデバイスドライバ

その実体はダイナミックリンクライブラリ

新たなユーザーインターフェースをもつプラットフォームとして，パソコンユーザーの注目と期待を集める MS-Windows 3.0．Windows アプリケーションからプリンタなどの周辺機器への入出力には，MS-DOS のデバイスドライバではなく，Windows デバイスドライバが必要であることは意外に知られていない．これは，各メーカーが自社製品を標準的なシステム構成（ディスプレイ，キーボード，マウス，プリンタなど）で使用する場合の各種 Windows デバイスドライバについては，Windows をインストールするだけで使用できるようにしていることによる．ここでは Windows のデバイスドライバの役割とその構造，デバイスドライバプログラミングに必要な各種ツールの解説や，プログラミングの手順など，Windows のデバイスドライバとそのプログラミングについての概要を紹介する．

Windows デバイスドライバの概要

Windows上で動作するアプリケーションプログラムは，キーボード，マウス，プリンタ，ディスプレイ，タイマ，RS-232Cポートなどをデバイスに依存せずに取り扱うことができる．これは，Windowsシステムがこれらのデバイスを一括して管理しくれるからだ．

Windows デバイスドライバの位置づけ

MS-DOSでは，アプリケーションプログラム，デバイスドライバ，周辺デバイスの関係は図1左のような概念図で示すことができる．アプリケーションプログラムが周辺デバイスに対して何らかの入出力を行おうとすると，MS-DOSが標準の，あるいはあとから組み込まれたデバイスドライバを呼びだす．デバイスドライバはソフトウェア割り込みによって，ROM BIOSにファンクションコールを行って，ROM BIOSが実際のアクセスを行う(ことになっている)．だが，実際はMS-DOSの機能はディスクアクセス以外はほとんどサポートしていないため，プログラムはMS-DOSを経由せず，直接ROM BIOSやハードウェアにアクセスしている．

これに対して，Windows上ではディスク以外のデバイスのアクセスはWindowsシステムが制御する．したがって，個々のWindowsアプリケーションが直接プリンタやマウスなどに命令を送る必要はない(図1右)．

Windows デバイスドライバの組み込み

もう少し詳しく見てみることにしよう．WindowsシステムはGDI，KERNEL，USERという3つのモジュールで構成されている．これらのモジュールはそれぞれGDI.EXE，KERNEL.EXE，USER.EXEという3つのファイルとして存在する．このうちKERNEL.EXEは，Windows 3.0ではリアルモード用，スタンダードモード用，エンハンストモード用の3種類がある．これらのファイルはインストール時にSYSTEMというサブディレクトリにコピーされてWindowsの起動プログラムWIN.COMがシステム状況に応じたファイルセットを起動するという仕組みになっている．

Windowsデバイスドライバモジュールは，これらの3つのシステムファイルとは別に存在し，対応するデバイスによってそれぞれ独自にファイル名をもつ．ファイル名は共通の拡張子.DRVをもつ．これらのWindowsデバイスドライバモジュールは，Windowsの起動時にWindowsの初期化ファイルSYSTEM.INIで定義された内容にしたがって，Windowsシステムに組み込まれるのである．SYSTEM.INIファイルの［boot］セクションで，次の形式で指定する．

表1　デバイスドライバモジュール

モジュール名	内容
COMM.DRV	RS-232Cドライバ
DISPLAY.DRV	ディスプレイドライバ
KEYBOARD.DRV	キーボードドライバ
MOUSE.DRV	マウスドライバ
NETWORK.DRV	ネットワークドライバ
SOUND.DRV	サウンドドライバ
SYSTEM.DRV	システムハードウェアドライバ

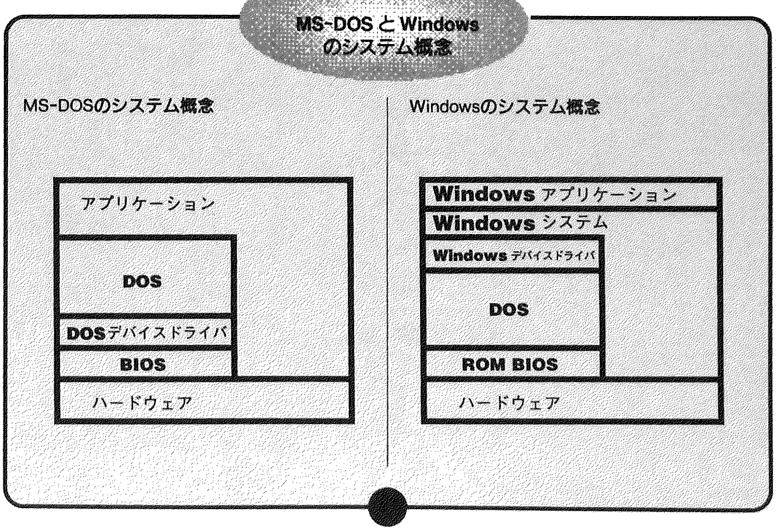


図1　MS-DOSとWindowsのシステム概念

モジュール名=<ファイル名>

バーチャルデバイスドライバ

Windowsには，バーチャルデバイスドライバというエンハンストモードの場合のみ必要な特殊なデバイスドライバが存在する．

Windows 3.0のエンハンストモードでは，画面上で複数のDOSウィンドウをオープンし，それぞれでMS-DOSアプリケーションを実行することができる．それぞれのアプリケーションが周辺デバイスに対して勝手に入出力命令を出してしまったのではまずい．これを制御するのが，バーチャルデバイスドライバの役割である．

図2は，エンハンストモード環境の概念図である．Windows 3.0のエンハンストモードは，80386/80486CPU上でDOSアプリケーションをマルチタスクで動かすしくみ，仮想マシンアーキテクチャを実現している．その中枢となるのはVMM(Virtual Machine Manager)で，メインメモリ，CPU実行時間，周辺機器など，このマルチタスク環境の実行に必要なさまざまなサービスを提供する．プリンタなどの周辺機器や，割り込みコントローラのような内部的なハードウェアデバイスに対する入出力が発生すると，VMMを経由して対応するバーチャルデバイスドライバに対してサービスが要求される．

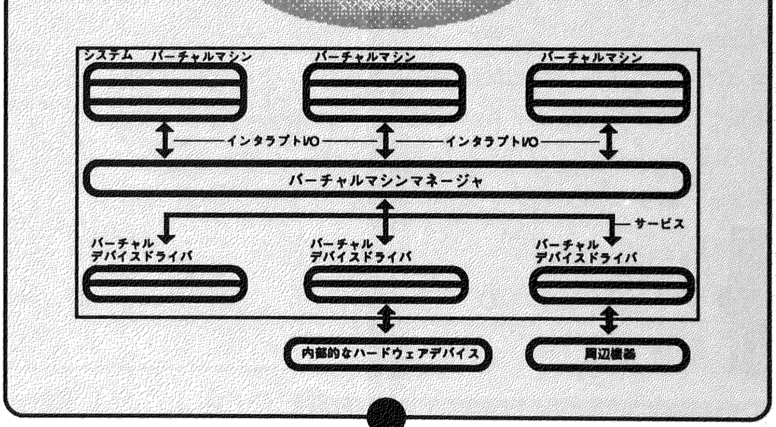
通常のデバイスドライバは，実際に円をプリントしたり，マウスの位置を取得したりといったデバイスの動きを司る，つまり抽象化されたデバイスAPIを実際のデバイスにあてはめるために存在している．これに対して，バーチャルデバイスドライバはエンハンストモードでデバイスへのアクセスに順序をつけて，仮想マシンによるマルチタスキングをサポートするために存在する．つまりデバイスに対するすべてのアクセスをトラップして，これを仮想化する役割を果たすわけだ．

バーチャルデバイスドライバの組み込み

バーチャルデバイスドライバは，Windowsがエンハンストモードで起動したときに，VMMとリンクされる．通常のデバイスドライバの場合と同じように，Windowsの初期化ファイルSYSTEM.INIで定義された内容にしたがって，Windowsシステムに組み込まれるのである．実際には，エンハンストモードでWindowsを実行するときの情報を示す[386Enh]セクションで指定する．

device =<ファイル名または＊デバイス名>

図2
バーチャルデバイスドライバの働き



COLUMN

Windowsを構成する3つのモジュール

・GDI.EXE

Graphics device interface．画面やほかの表示デバイス上にイメージを表示するのに必要なグラフィック操作を行う．

・KERNEL.EXE

Windowsで使用するすべてのマシンリソースを制御し，わりあてる．OSとともにメモリを管理し，アプリケーションをロードし，現在実行中のプログラムやほかのタスクをスケジューリングする．

・USER.EXE

画面上にウィンドウを表示する．ウィンドウの生成，サイズの変更，移動，破壊などのユーザーリクエストを実行し，アイコンやカーソルの制御，アプリケーションに対するキーボードやマウスからの入力を制御する．

=の後ろには，仮想デバイスのファイル名,またはアスタリスク(*)に続けてデバイス名を直接指定する.バーチャルデバイスドライバのファイル名の拡張子には「386」が使われる.*に続けて指定されるデバイス名は，エンハンストモードの仮想マシンマネージャである WIN386.EXE中で指定される仮想デバイスを指定しなければならない.

デバイスドライバを作るには?

Windowsデバイスドライバは，MS-DOSのデバイスドライバとは，その位置づけや果たす役割が違う．Windowsデバイスドライバのプログラミングには,DOSでの知識はほとんど役にたたない．前提として求められるのはWindowsプログラミングに関する一般的な知識である.

Windows デバイスドライバの構造

Windowsデバイスドライバの実体は，Windowsの重要な構成要素のひとつであるDLL (Dynamic - link Library) である．DLLはプログラムの実行時にリンクされるライブラリルーチンである.

通常MS-DOS上でプログラムを開発する場合には，必要なモジュールはあらかじめすべてリンクしておく必要がある．この方式はスタティックリンクと呼ばれる．これに対して，DLLは実行時にはじめてプログラムとリンクされる．この方式はダイナミックリンクと呼ばれる.

Windowsシステムそのものも，このDLLを利用している．Windowsを構成する3つのファイル，GDI.EXE，USER.EXE，KERNEL.EXEなどもすべてDLLである.Windowsの標準的な関数は，これら3つのDLLのいずれかに含まれているため，複数のアプリケーションが動作しているときでも，使用している関数が同じなら，同じコードが共有される.

DLLとアプリケーションの関係

Windowsの基本的な構成単位となるのは「モジュール」である．通常Windowsアプリケーションの実行ファイルは1モジュールである．モジュールには，アプリケーションモジュールと先にのべたDLLモジュールの2種類がある．アプリケーションモジュールであるか,DLLモジュールであるかは，ファイル名や拡張子からは判断できない．Windowsのシステムファ

リスト1　SYSTEM.INIファイルの[386Enh]セクションの例

```
[386Enh]
ebios=
display=vddn.386
keyboard=*vkd
mouse=*vmd
network=*vnetbios, *dosnet
device=*vpicd
device=*vtd
device=*reboot
device=*vdmad
device=*vsd
device=*vsound
device=v86mgrn.386
device=*pageswap
device=*dosmgr
device=*vmpoll
device=*wshell
device=*vfd
device=*vscsid
device=vpdn.386
device=*parity
device=*biosxlat
device=*vcd
device=*vmcpd
device=*combuff
device=*vhd
local=CON
FileSysChange=off
MinTimeslice=20
WinTimeslice=100,50
WinExclusive=0
Com1AutoAssign=2
LPT1AutoAssign=60
Paging=Yes
GRAB386.FON=GRAB386.FON

```

COLUMN

SYSTEM.INIファイル

Windows 3.0には，WIN.INI，SYSTEM.INIの2種類の初期化ファイルが存在する．WIN.INIはおもにWindowsやアプリケーションのソフトウェア環境を，SYSTEM.INIはおもにフォントやデバイスドライバなどの設定を保存するファイルである.

一部のデバイスに関しては，Windowsのインストールプログラムを起動して設定を変更するだけで，自動的にSYSTEM.INIファイルが書き換えられる.

SYSTEM.INIファイル中の各セクションやディレクティブに関する説明は，SYSINI.TXT，SYSINI2.TXT，SYSINI3.TXTという3つのテキストファイルにおさめられている．SYSTEM.INIファイルは，MS-DOSのテキスト形式のファイルなので，Windows付属のメモ帳や各種エディタなどで編集可能である．ただし誤って変更すると，Windowsシステムそのものが機能しなくなるおそれがあるので，変更には注意を要する.

イルであるGDI.EXE, USER.EXE, KERNEL.EXEはEXEファイルであるが，単独では実行できないDLLモジュールである．両者の違いは，形式ではなく，Windowsのタスクとして実行できるかどうかという点にある．

Windows上のモジュールはダイナミックリンクを実現するために，そのモジュール内の関数で外部のモジュールから呼び出すことができるものは何か，そのモジュール内で呼び出す外部モジュールの関数は何かを明示しなければならない．前者をエクスポート（EXPORTS）後者をインポート（INPORT）という．DLLは，アプリケーションやほかのDLLがそのライブラリ内の関数を使えるように，関数名をエクスポートする．一方，DLLを利用するアプリケーションの側では，利用したいDLL内部の関数名をインポートするのである．

アプリケーションモジュールとDLLモジュールは，見かけ上は同じ形式をもつ．この形式はニューEXEヘッダフォーマットと呼ばれる．ニューEXEヘッダフォーマットは，プログラムのダイナミックリンクを可能にするため，ヘッダにモジュールに関する情報を保持している．この情報はMS-Cに付属するEXEHDRユーティリィを使って読むことができる．EXEHDRを使えば，そのモジュールがアプリケーションなのか，DLLなのかを知ることができる．

プログラマから見た場合，アプリケーションとDLLのもっとも大きな違いはスタックに関するものだ．DLLモジュールはそれ自身でスタックをもたない．DLLを呼び出したタスクのスタックセグメントを利用するのである．Cランタイムライブラリルーチンでは，DSとSSの値が同じであることを前提とするものが多いので，これらの関数をDLL中で利用するときには注意しなければならない．

デバイスドライバを作るのに必要なもの

Windowsデバイスドライバを作るには，通常のWindowsアプリケーションの作成に必要なものと同じハードウェア，ソフトウェアが必要である．これに加えてデバイスドライバ開発キット（DDK）が必要となる．

●●● ソフトウェア

DDK（Device Development Kit）

本誌2月号でも簡単に紹介したが，MS-Windows用のデバイスドライバを書くには必須のパッケージ．アメリカでは，Windows 3.0用から一般ユーザーでも購入できるようになった．

IBM PC版のDDKには，ビデオ，プリンタ，キーボード，シリアルポート，マウス，サウンドなど各種のデバイスについてWindows用デバイスドライバを書くためのドキュメント，各種のサンプルソースプログラム，開発ツール，テストツールなどが含まれている．また，Windows 3.0で標準にサポートされているデバイスドライバのソースコードがすべて含まれている．

この原稿を書いている91年5月現在，日本ではWindows 3.0用DDKは市販されていない．今後，市販することになるのかどうかについても，現状ではまだはっきりしていないという．

Windowsソフトウェア開発キット（SDK）

Windowsプログラム用の開発ツール集．Windowsプログラム用の各種ヘッダファイル，ライブラリ，ユーテ

COLUMN

リアルモード，プロテクトモード，仮想8086モード

インテルの80286以上のCPUには，リアルモードとプロテクトモードという2種類の動作モードが存在する．80286以上のCPUを実装したマシンでMS-DOSを使って各種アプリケーションを実行するときは，通常はリアルモードで動かすことになる．これは，CPUを高速8086として利用しているに過ぎない．

これに対して，プロテクトモードでは1MB超のメモリ空間にアクセスが可能で，ディスクリプタテーブルを使ってメモリを管理，保護する仕組みをハードウェア的に備えている．80386以上のCPUでは，オフセットアドレスを32ビットで表現できるので，セグメントの切り替えなしに4GBのメモリ空間を利用することが可能である．さらに，80386/486CPUでは，プロテクトモードのサブ・モードである仮想8086モードが利用できる．これは，80386/486 CPUのプロテクトモード中で複数の8086マシン環境（仮想86マシン）を作り出す機能で，CPUはそれぞれの仮想86マシンをタスクとして認識する．実際にこの機能を使ってMS-DOSのアプリケーションプログラムを複数実行するには，CPUのモードを切り換えて，周辺デバイスを制御したりするソフトウェア上の仕組みが必要である．

ィリティを含む．リソースコンパイラ（RC），SDKペイント，ダイアログエディタ，デバッガなどのツール類がおさめられている．

Windowsデバイスドライバの作成には，Windowsアプリケーションと同じくWINDOWS.Hなどのヘッダファイルなどが必要．また，CodeView，SYMDEB，WDEB386などのデバッガも必要だ．さらに，デバイスドライバ中でダイアログボックスなどのリソースを使用する場合は，リソースコンパイラ，リソースエディタなどのツールも必要となる．

MASM

Ver.5.1以降のMicrosoft Macro Assembler（MASM）．

Cコンパイラ

プリンタドライバの記述には，MS-C Ver.5.1以降のCコンパイラが必要となる．

これらの開発を行ったり実際に動かす環境として，MS-DOS(Ver.3.1以降)，MS-Windows（Ver.3.0）が必要なのはいうまでもない．

●●● ハードウェア

パソコン本体，ハードディスク，カラーモニタ，マウスは必須．これ以外の機器についてのドライバを開発する場合は，対象となるデバイスが必要なのはもちろんである．上記のソフトウェアをすべてインストールすることを考えると，ハードディスク容量は多いに越したことはない．100MB程度の容量があることが望ましい．

デバイスドライバの作成手順

Windowsデバイスドライバの作成手順は，Windowsアプリケーションの作成手順に準ずる．図3は一般的なWindowsアプリケーションの作成手順である．ただし，Windowsデバイスドライバは DLL であるため，アプリケーションとは多少異なる点もある．

●●● コンパイルとアセンブル

C言語で記述されたDLLのソースプログラムをコンパイルするときには，コンパイルオプションとして次のものが必要となる．

−Gw

エクスポートされた関数にWindowsのプロローグコード，エピローグコードを追加．

−Aw

ポインタが32ビットにキャストされたときに，適切なセグメントアドレスが補充されるようにする．

デバイスドライバは，アセンブラで書かれることが多い．アセンブラの場合はDDKに含まれる次の3つのファイルをインクルードするのが必須条件である．

CMACROS.INC

CMACROS.INCにはWindowsアプリケーションの作成に必要なセグメント，プログラミングモデル，関数インターフェース，データ型などを定義する各種のマクロが含まれる．また，このファイルをインクルードするとアプリケーションのメモリモデルと呼び出し規約を定義するアセンブル時オプションを指定できるようになる．実際にはアセンブラソースコードにリスト2のように記述する．

GDIDEFS.INC

GDIDEFS.INCをインクルードするとWindowsの標準的な名前の記号定数や構造体が使えるようになる．GDIDEFS.INCの一部分だけを選択して定義することもできる．

WINDEFS.INC

WINDEFS.INCには，マクロEnterCritとLeaveCritが含まれている．このマクロを使うと，フロッピーやハードディスクコントローラ，数値演算コプロセッサ，タイマ，キーボード，マウスなどの非同期のハードウェア割り込みによって，現在実行中の処理が妨げられるのを防ぐことができる．

なお，これらのファイルの中にはCとアセンブラの両方の定義が含まれるので，C言語でデバイスドライバを書く場合にも使える．プリンタドライバを作成するときには，WINDOWS.H，場合によってはSPOOL.Hも必要である．

●●● リンク

Windowsアプリケーションや DLLのリンク時には，Windowsのモジュールに関する情報をモジュール定義ファイルで指示する必要がある．モジュール定義ファイルの内容はアプリケーションの場合とDLLの場合では異なる．DLLを作成する場合，モジュール定義ファイル中に次の文は必ず含めなければならない．

●●● LIBRARY文

ダイナミックリンクライブラリのモジュール名を指定する．アプリケーションモジュールの場合は，LIBRARY文の代わりにNAME文が含まれる．

●●● EXPORTS文

エクスポートされる関数を指定する。この文で指定した関数がほかのアプリケーションやDLLからインポートできるようになる。DLLの場合は関数名の後ろにオーディナリ番号がつく。

●●● EXETYPE WINDOWS文

この文によって，リンカはDLLにWindowsのフォーマットヘッダを付加する。アプリケーションモジュール，DLLモジュールともにこの文は必須。

Windows デバイスドライバの実際

その1 プリンタドライバ

プリンタドライバのおもな役割は，デバイスから独立したGDIのインターフェースを，その機種に依存した一連のプリンタコマンドやデータに変換することである。

Windows環境での印刷のしくみ

Windows環境では，アプリケーションが直接プリンタに命令を送るわけではない。アプリケーションは，出力をプリンタのデバイスコンテキストに対して送るだけでよいため，個々のアプリケーションがプリンタのハードウェア的な相違を意識する必要はない。デバイスコンテキストとは，ディスプレイやプリンタのようなさまざまな出力デバイスに対する描画情報を定義するものである。

図4はアプリケーションの出力がアプリケーションからデバイスコンテキスト，デバイスドライバを経由して実際に印刷されるまでを示している。Windowsアプリケーションがデバイスコンテキストを作成したり，デバイスコンテキストに処理を要求したりするときには，GDI (Graphics Device Interface)関数と呼ばれる各種の関数を使う。GDIは，アプリケーションからの関数呼び出しを，デバイスドライバの呼び出しに変換する。デバイスドライバは，さらにこれをそれぞれのプリンタの種類に応じた処理に変換し，この個々のプリンタ機種固有の処理をプリントマネージャに転送する。プリントマネージャは，アプリケーションから転送されたコマンドをためこんで，通信ジョブとし，これをプリンタに転送する。

出力までにこのように複雑な経路をたどることによって，複数のアプリケーションが同時に動作し，それらが同時に出力要求を出した場合でも，Win-

図3 Windowsアプリケーション作成手順，Cの場合，アセンブラの場合

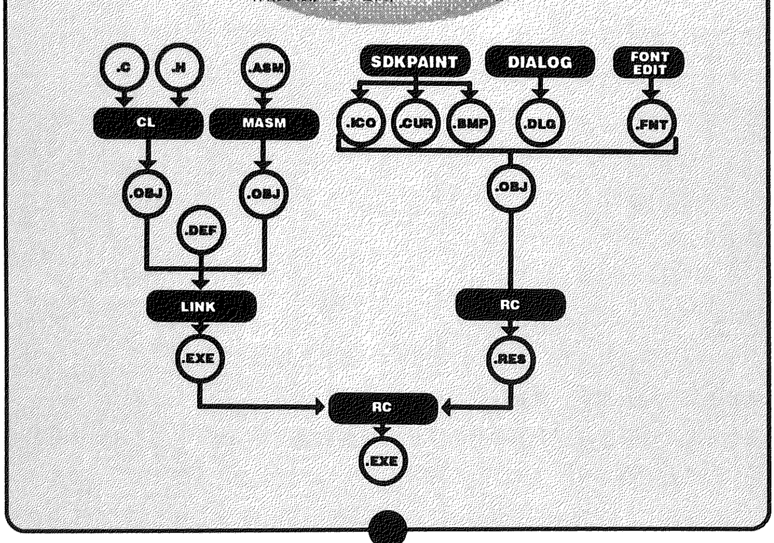
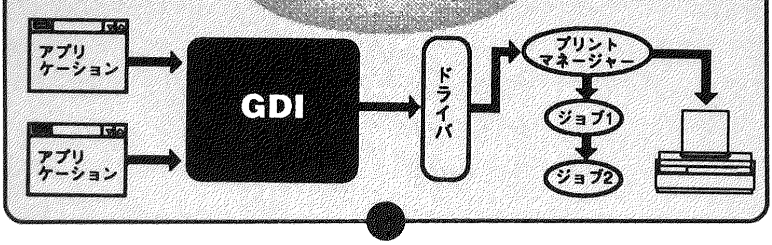


図4 プリンタへの出力の流れ



リスト2 CMACRO.INCの使用例

```
memS                    ;use small model
?PLM=1                  ;use Pascal calling
?WIN=1                  ;generate prolog-epilog code
?CHKSTK=1               ;call CHKTSK for all procs in this file
include CMACROS.INC


```

dowsはアプリケーションごとに区別して出力してくれる．

Windows プリンタドライバの構造

では，実際のプリンタドライバの構造はどうなっているのだろうか？日本語のDDKがまだ一般には提供されていない以上，詳しい機種別ドライバの内容に触れることはできないのだが，Windowsのプリンタドライバの開発に携わっているイースト(株)の小松直樹主任によれば，プリンタドライバは，①GDIとのインターフェース関数②アプリケーションプログラムとのインターフェース関数の集まりであるという．「プリンタデバイスドライバはWindowsのDLLです．あらかじめ20ほど関数の名前が用意されていて，それに基づいて実際のプリンタのハードウェアに応じて関数の中身を変えていくという形になります．たとえば，プリンタへの出力部分などは，プリンタごとに少しずつ違いますね．それに合わせて書き換えていくことになります」(小松氏)．

Windowsプリンタドライバの満たすべき機能やこれが呼び出される手順は，Windowsシステムであらかじめ定められており，これはどのプリンタドライバでも共通である．実際のプリンタのもつ機能に見合う形で関数をインプリメントするのがプリンタドライバの作成作業ということになる．

プリンタドライバ作成における諸問題

Windows GDIで定められた関数群をインプリメントするだけでよいのなら，DDKさえ手にはいれば，プリンタドライバのプログラミングは簡単にできるのだろうか？ 前出の小松氏によれば，プリンタドライバのプログラミングにおいては関数をインプリメントすることより，その後のチューンナップがたいへんなのだという．

たとえば，図形描画能力の低いシリアルプリンタなどの場合，バンディングと呼ばれる手法を使ってグラフィックスを処理する．GDIのシミュレート機能を使ってグラフィックスデータをメモリビットマップ上に展開してから出力するのだが，このときメモリの制限から1ページ分のデータを一度に処理できない．そこで，1ページを帯状にいくつかの部分に分けて処理するのである．しかし，テキストの印刷にまでこの手法を使うと，処理速度が遅くなってしまう．そこでバンディングをサポートしているドライバでも，テキスト印刷にはプリンタのフォントをそのまま使うことがある．ところが，グラフィックスはビットマップ処理，テキストはプリンタフォントということになると，グラフィックスとテキストを重ね合わせることができない．

では，図形描画能力の高いページプリンタやPostScript対応プリンタなら問題はないかというと，今度はプリンタの描画能力そのものが，GDIの要求するラスタオペレーションをすべて実現できるかという問題になる．

結局のところWindowsのソフトウェアが要求するだけの高品質のプリンタは現実には存在しないし，GDIの仕様をすべて満足させることのできるプリンタドライバもまたあり得ない．実用的な速度と印字品質をどこまで実現できるか，どう見極めをつけるかがプリンタドライバ作成の最大の難関だといえる．

Windows デバイスドライバの実際 その2 ディスプレイドライバ

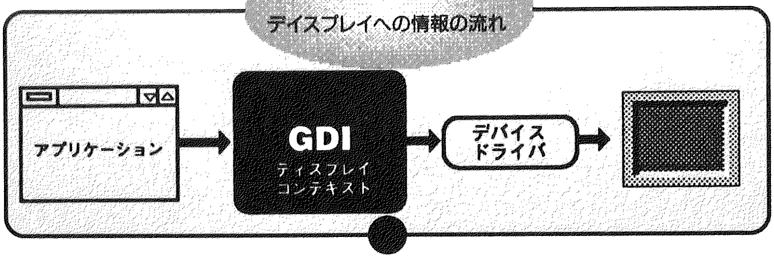
Windowsディスプレイドライバは，通常Windowsのパッケージの中に含まれているものをインストールするだけでよく，プリンタドライバのようにユーザーがそのドライバを使っていることを意識しなければならないという性質のものではない．しかし，ディスプレイドライバはWindowsシステムの画面表示には欠くことのできないモジュールである．

Windows 環境での画面表示のしくみ

すべてのWindowsアプリケーションは，画面に文字やイメージを表示することができるが，複数のアプリケーションが同時に動作可能な環境であるため，それぞれのアプリケーションが自由に画面全体を書き換えることはできない．Windowsは画面表示全体を管理するために，デバイスコンテキストの一種で，画面に表示されたウィンドウをそれぞれ独立したディスプレイ画面として扱うためのしくみであるディスプレイコンテキストを使用する．

図5はアプリケーションの出力がアプリケーションからディスプレイコンテキスト，ディスプレイドライバを経由して画面表示されるまでを示している．Windowsアプリケーションが画面に文字やビットマップを表示するGDI関数を使用すると，GDIは，アプ

図 5
ディスプレイへの情報の流れ



リケーションからの関数呼び出しをディスプレイドライバの呼び出しに変換する. ディスプレイドライバでは, 画面への出力を制御する.

Windows ディスプレイドライバの役割

基本的にはディスプレイドライバもディスプレイの種類ごとに必要である. だが, NECのPC-9800シリーズのように, 共通となるアーキテクチャをもっている場合 (VRAMのアドレスやGDCなど)には, 同じデバイスドライバが利用できる. ハイレゾリューションモードでディスプレイ表示をする場合などには, 標準のものとは別のディスプレイドライバを使うことになる.

Windowsシステムでは, ディスプレイ上にタイトルバーやスクロールバーなどをもつウィンドウを表示する. スクロールバーの形状を決める処理などは, Windows GDIで行われるが, 実際のタイトルバーやスクロールバー表示, 画面上に線を引いたりという指示は, GDIからディスプレイドライバに送られてくる. これらの処理を実現するのがディスプレイドライバの役割だ.

ディスプレイドライバの構造

ディスプレイドライバは, GDIから呼び出される関数の集まりである. プリンタドライバの場合とは違って, アプリケーションから直接呼び出されることはない. Windowsディスプレイドライバの満たすべき機能やこれが呼び出される手順は, Windowsシステムであらかじめ定められている. 対象となるディスプレイのもつ機能に見合う形で関数をインプリメントするという手順はプリンタドライバの場合と変わらない.

ディスプレイドライバ作成における諸問題

「ディスプレイドライバのプログラミングでも, インプリメントして動かすまではそうたいへんではありません. むしろ, そこからいかにチューンナップ, スピードアップするかです」と語るのは, Windows 3.0対応のディスプレイドライバを開発した経験をもつ株式会社エム・ピー・テクノロジーの若月文男氏.「ディスプレイドライバの場合は, プリンタドライバ以上にスピードが重視されます. 表示が遅ければ, Windowsそのものが遅いということにつながってしまうからです. それに, 動作がおかしければ, すぐ異常が目に見えてしまいます.」

ディスプレイドライバはプリンタドライバと比べると, よりシステムに近いドライバである. そのため, インプリメントには別の問題もあったという. IBM PC対応のDDKには, CGA, EGA, VGAなどIBM PCの規格にあうディスプレイドライバのサンプルソースコードが含まれている. ところが, 画面の解像度などを示す肝心のパラメータに関する説明がなく, 解像度の違う国産PC用のディスプレイドライバの作成では苦労したのだそうだ.

以上Windowsデバイスドライバについて, 大まかな概要を見てきた. 折しも先日, 三菱電機がWindows 3.0用に自社のカラープリンタのデバイスドライバを74,000円で発売することを発表した. この価格は同じく三菱電機がMAXY用に発表したWindows 3.0の約3倍にあたる. デバイスドライバを個人で開発する環境が整わない以上, いまのところ各ハードウェアのメーカーが提供してくれるのを待つ以外に手はない. 数はまだまだ少ないがWindowsアプリケーションの開発を手がけるプログラマも現れてきた. 早くDDKを一般にも販売する体制を整えてほしいものだ.

(編集部)

[参考資料]
「プログラミング WINDOWS」(CHARLES PETZOLD 著, エー・ピー・ラボ訳, アスキー出版局)
「MS-WINDOWS プログラマーズガイド」
「MS-WINDOWS プログラミングツールズ」
「MS-WINDOWS プログラマーズリファレンス」(Vol.1〜5)(以上日本電気株式会社)
「Microsoft Windows Device Development Kit」(米 Microsoft 社)

[協力]
イースト株式会社
株式会社エム・ピー・テクノロジー

特集
最新「デバイスドライバ」研究



OS/2のデバイスドライバ

マルチタスク環境下のデバイス制御

OS/2は，米国マイクロソフト社とIBM社が共同で開発した80286，80386上で動作するマルチタスク，シングルユーザーのオペレーティングシステムである．一時期はMS-DOSの次世代OSとして注目されたものの，最近はWindows 3.0の影に隠れてOS/2への関心は低くなっている．しかし，IBM社が，MS-DOS，Windows，OS/2，OS/2 PMのすべてのソフトウェアを動作させる共通プラットフォームとしてOS/2 2.0を据えた戦略を発表するなど，OS/2が再び重要な存在となる状況もありえる．ここでは，OS/2のデバイスドライバについて，その概念と構造についてMS-DOSとの比較を中心に解説する．

OS/2の特徴

プレゼンテーションマネージャ(PM:Presentation Manager)

PMは，OS/2 1.1から付属しているGUI (Graphic User Interface)である．見かけや操作方法はほとんどMS-Windowsと同じである．

新しいAPI (Application Program Interface)

MS-DOSでファンクションコールと呼ばれている機能に匹敵するもので，OS/2のカーネルが約250個，PMが約500個のサービスを提供している．

これらのAPIはfarコールで起動される．パラメータはスタック上で渡され，呼び出し側のメモリ領域中の変数に値が返される．アドレスの受渡しは，farポインタで行われる．

また，OS/2のAPIは，高級言語から直接アクセスすることが可能である．

プロテクトモードのサポート

MS-DOSマシンが採用しているインテル社の80系のプロセッサは，リアルモードにおいてセグメントアドレスとオフセットアドレスを組み合わせて物理アドレスを得ている．リアルモードでアクセスできるメモリは最大1MBで，これがMS-DOS上のプログラミングにおける大きな足かせとなっていた．

OS/2は，80286以上のプロセッサのもつプロテクトモードで動作し，最大16MBのメモリに直接アクセスすることができる．

プロテクトモードにおけるメモリアクセスは，ディスクリプタテーブルを用いて間接的に行われる．ディスクリプタテーブルには，メモリセグメントのベースアドレスとその大きさ，属性，特権情報が含まれている．プロテクトモードにおいて，セグメントレジスタの内容はセレクタと呼ばれ，ディスクリプタテーブルへのインデックスとして働く．メモリの参照は，セレクタの示すディスクリプタテーブル内のベースアドレスとオフセットが組み合わされて作られる24ビットの物理アドレスによって行われる．

ディスクリプタテーブルには，1つのグローバルディスクリプタテーブル(GDT)と各プロセスごとのローカルディスクリプタテーブル(LDT)の2種類がある．

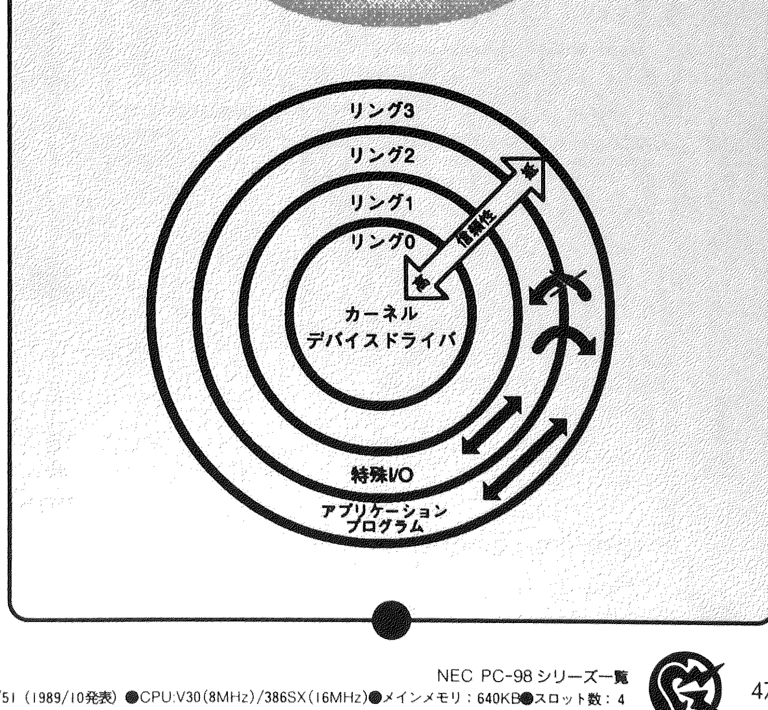
また，プロテクトモードには，特権レベルと呼ばれる機能が存在している．これはリングプロテクションとも呼ばれ，図1に示す0から3までの4つのレベルがある．CPUは，ディスクリプタテーブルと特権レベルの機能を用いてメモリの保護を実現している．

仮想記憶のサポート

OS/2は，実際に存在するよりも多くのメモリの使用を可能にする仮想記憶をサポートしている．80286では，1GBの仮想記憶が使用可能である．

プロセスがメモリを要求すると，VMM (Virtual Memory Manager)はセグメントを移動して物理メモリの未使用ブロックをひとつにするか，読み込み専用のデータセグメントやテキストセグメントを破棄して要求を満たそうとする．これらの方法が失

図1 リングプロテクション



敗すると，VMMはスワッパ(Swapper)を起動して1つ以上のセグメントをディスク上にスワップアウトする．スワップアウトされたセグメントのセレクタを使用したメモリアクセスが行われるとハードウェアトラップが発生し，VMMが必要なセグメントをディスク上のスワップファイルか.EXE，.DLLファイルから物理メモリに読み込む．

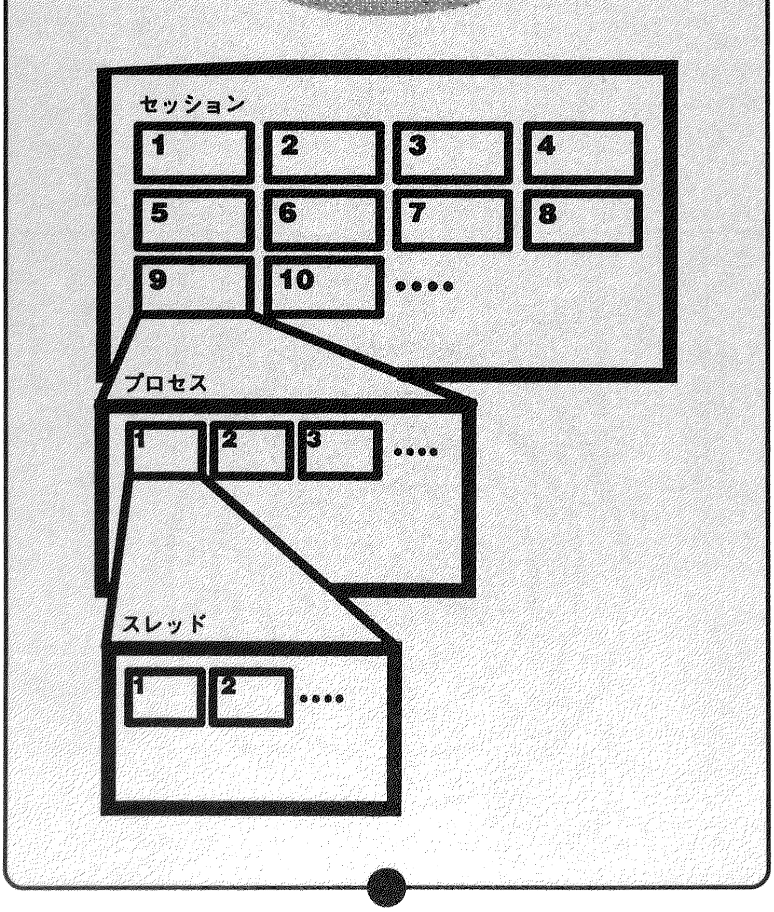
プリエンプティブなマルチタスク

マルチタスクOSにおけるタスク間のCPU時間の割り当てを管理する部分をスケジューラ，1つのタスクから別のタスクへの切り替えをコンテキストスイッチと呼ぶ．マルチタスクOSには，基本的にイベント駆動型(event-driven)のスケジューラを用いるものとプリエンプティブ(preemptive)なスケジューラを用いるものの2つがある．

OS/2が採用しているのはプリエンプティブなスケジューラであり，一定の間隔で生成される外部信号（クロックデバイスなどのハードウェア割り込み）によって，ある時間ごとに強制的にタスクの実行を中止し，次のタスクを実行させる．

OS/2のタスクの基本概念には次の3つがある（図2）．

図2
OS/2タスクの基本概念



セッション

OS/2上でユーザーが使用するハードウェア（ディスプレイ，キーボード，マウス）をスクリーングループといい，これに対応するプログラムをセッションと呼ぶ．セッションはセッションマネージャによって管理され，セッションマネージャはスクリーングループの生成と消滅を行い，これにプログラムを対応させる．

プロセス

プロセスとは，ロードされたプログラムとそのプログラムが所有しているすべての資源を指す．つまり，OS/2はプロセス単位で資源の管理を行っているのである．プロセスが所有する資源の情報には，プロセスID，スワッピング情報，LDTポインタ，プロセス間通信の情報，デバイスモニタ，メモリ，セマフォ，子プロセス情報，プリンタなどのシステム資源，などがある．プロセスは子プロセスを生成することができ，子プロセスは親プロセスの資源を一部継承する．

スレッド

OS/2で実際にCPU時間を割り付けられる実行の基本単位をスレッドと呼ぶ．スレッドは，スレッドID，スレッドの状態，プライオリティ（優先度），CPUレジスタ値，スタックポインタなどの情報をもち，Blocked（ブロックしている），Ready（実行可），Running（実行中）のいずれかの状態にある．1つのプロセス内で複数のスレッドの実行が可能で，1つのシステム内で複数のスレッドと複数のプロセスの並列処理ができる．タスク切り替え時には，Ready状態のスレッドの中からもっともプライオリティの高いものが実行される．

NEC PC-98シリーズ一覧
PC-H98model70（1989/10発表）●CPU:80386(8/20/33MHz)●メインメモリ：1.6(5.6)MB●スロット数：4（NESA）●備考：ハイレゾモデル

プロセス間通信機能

OS/2は，マルチタスクで動作するプロセスおよびスレッドの間で情報を交換するメカニズムとしてプロセス間通信（IPC：Inter Process Communication）機能をもっている．

IPC機能はOS/2のAPIとして提供され，プログラマはファンクションコールによって排他制御や同期処理を実現することが可能となる．

OS/2が提供するIPC機能には，次の5つがある．

セマフォ

セマフォは，セット（所有している）とクリア（所有されていない）という2つの状態をフラグ方式で表すことができる簡単で高速な通信機能である．

OS/2のセマフォには，RAMセマフォ，システムセマフォ，Fast-Safe RAMセマフォ（FS RAMセマフォ）の3つがある．

RAMセマフォは，同一プロセス中の複数のスレッド間やメモリを共有する複数のプロセス間で通知を送ったりリソースを制御したりするもので，システムセマフォはプロセス間でのシグナル発行や同期に使える名前付きのオブジェクトである．FS RAMセマフォは，RAMセマフォとシステムセマフォの両方の特徴を備えたもので，複数のクライアントがある場合にDLLによって使われる．

パイプ

パイプは，プロセス間で任意の大きさ（最大で64KB）のデータをFIFO（First In First Out）方式でやり取りする機能である．パイプには，名前なしパイプと名前付きパイプの2種類がある．

名前なしパイプは，資源を継承している親子関係にあるプロセス間でしか利用できず，またデータをストリームとして一方向に送ることしかできない．

名前付きパイプは，その名前によってシステムのどこからでもアクセスすることができ，データを双方向にやり取りできる．また，ネットワークを介したデータ転送も可能である．

キュー

キューは，すべてのプロセスがオープンしたり書き込みできる名前付きのオブジェクトで，そのデータ量は仮想メモリのみに制限される．キュー中のレコードは，FIFOでもFILO（First In Last Out）でも，また優先度によっても順序を決めることができる．

共有メモリ

共有メモリは，2つ以上のプロセスが同じセグメントのセレクタをもつことで高速に大量のデータ交換を行うメカニズムである．この機能を実現するために，OS/2は名前付きセグメントの生成とセレクタの供与・取得という2つの方法をサポートしている．

共有メモリは単にプロセス間で同じメモリセグメントを共有しているだけなので，ほかのIPC機能を利用して排他制御や同期処理を行う必要がある．

シグナル

シグナルは，ハードウェア割り込みによく似た機能である．外部イベントが生じると，シグナルはシグナルを利用するハンドラをもつプロセスに非同期に通信を行う．

シグナルのハンドラをもつプロセスは，シグナルが送られたなら必ずこれを受け取らなければならない．制御は，シグナルハンドラとして指定されたルーチンに即座に渡され，ハンドラが処理を完了してリターンすると割り込みの入った位置に戻る．

ダイナミックリンク

MS-DOSでアプリケーションプログラムを作成する際，ライブラリルーチンへの外部参照は，リンク時に解決される．このようなリンクをスタティックリンクと呼ぶ．これに対してOS/2におけるライブラリルーチンへの外部参照は，アプリケーションをメモリにロードまたは実行した時点で解決される．このような外部参照の方法をダイナミックリンクといい，このとき参照されるライブラリをダイナミックリンクライブラリ（DLL：Dynamic Link Library）という．

ダイナミックリンクを使うと同一のライブラリルーチンを複数のアプリケーションが共有することが可能になり，メモリを節約することができる．また，アプリケーションにライブラリコードが組み込まれないため実行ファイルが小さくなり，ロード時間が短縮される．

OS/2のカーネルやPMで利用できる多くのファンクションが，ダイナミックリンクライブラリの形で供給されている．

DOS互換モード

OS/2は，従来のMS-DOSアプリケーションをプロテクトモードアプリケーションと並行して動作させるDOS互換モードを備えている．

OS/2はMS-DOSファンクション

コールをトラップし，OS/2のAPIファンクションコールに変換する．また，入出力などを実行する際には，リアルモードとプロテクトモードを交互に切り替える．

DOS互換モードは，かなりハードウェアに依存したMS-DOSアプリケーションに対しても有効だが，完全というわけではない．また，このDOS互換モード内でMS-DOSアプリケーションがハングアップした場合，システム全体がクラッシュする可能性が高い．

OS/2の基本構造

OS/2のシステムは，デバイスドライバ，カーネル，サブシステム，コマンドプロセッサ，PMからなる（図4）．

●●● デバイスドライバ

カーネルとデバイス（周辺機器）の間に存在して，カーネルを個々のハードウェアから独立させ，入出力を制御するシステムプログラム．

●●● カーネル

カーネルは，OS/2が提供する基本的なサービス（プロセス管理，メモリ管理，プログラム管理，タイム管理，入出力管理など）を実現するものである．

アプリケーションは，OS/2のAPIファンクションを通してカーネルの機能にアクセスする．

●●● サブシステム

サブシステムは，必要に応じてロー

図3
ダイナミックリンク

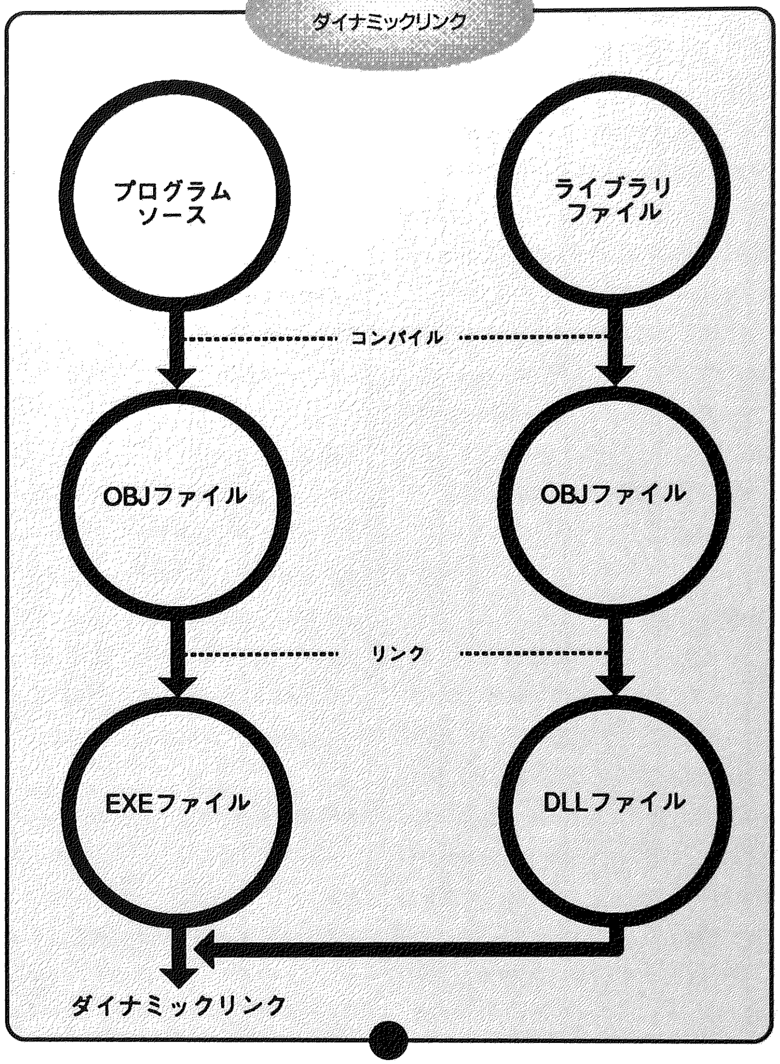
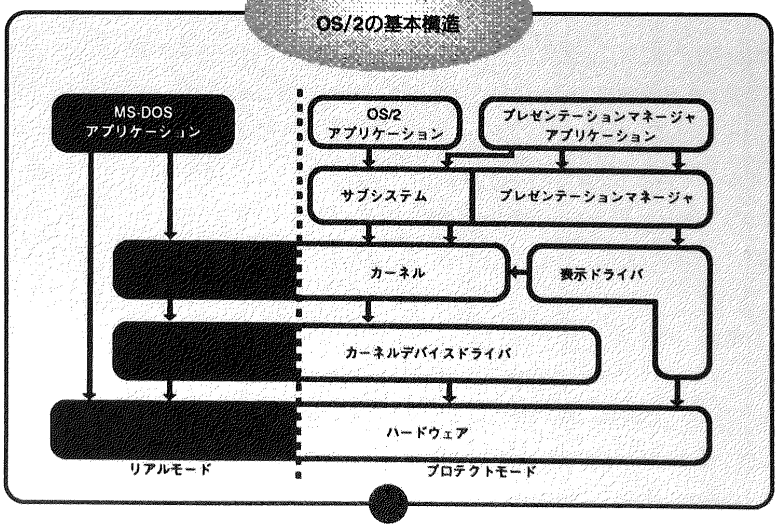


図4
OS/2の基本構造



ドされるサブルーチンを含んだダイナミックリンクライブラリである．

コマンドプロセッサ

コマンドプロセッサは，OS/2システムに対してキャラクタベースのユーザーインターフェースを提供している．デフォルトのコマンドプロセッサは，プロテクトモードではCMD.EXE，リアルモードではMS-DOSと同様COMMAND.COMである．

CMD.EXEはユーザーモード（特権レベルのリング3）で動作し，並行して複数のコピーを実行できる．

PM

PMについては，すでに「OS/2の特徴」で述べているので割愛する．

OS/2デバイスドライバの特徴

OS/2のデバイスドライバもMS-DOSのデバイスドライバも，デバイスとOSの間にあってデータの入出力制御を行い，アプリケーションのデバイス独立を保つという点はまったく同じである．しかし，OS/2のデバイスドライバにはMS-DOSのデバイスドライバにはない機能が多く含まれている．

次にOS/2のデバイスドライバ特有の機能をあげる．

マルチタスクへの対処

OS/2はマルチタスクのOSなので，複数のプロセスが並行して動作し，それぞれのプロセスが別々に入出力要求を出す複数のスレッドをもつ．このため，OS/2のデバイスドライバはリエントラント（再入可能）でなければならず，また同一デバイスに対する入出力要求を管理できなければならない．

さらに，OS/2のデバイスドライバは特権レベルのリング0で動作しているためカーネルからタスクを切り替えることができない．このため，デバイスドライバは自分自身で速やかにカーネルにリターンし，CPUをほかのスレッドに渡す必要がある．

デバイスヘルパルーチン

OS/2のデバイスドライバは，入出力要求を処理している最中にカーネルを呼び出してそのサービスを受けることができる．ただし，デバイスドライバは特権レベルのリング0で動作しているため，ダイナミックリンク形式のAPIを利用することはできない．なぜなら，APIはリング3で動作するプログラムからリング0で動作するシステムを呼び出すように設計されているからである．このため，デバイスドライバ専用のシステムサービスとしてデバイスヘルパルーチンが用意されている．

バイモーダル処理

OS/2ではDOS互換モードでMS-DOSアプリケーションを実行できるため，OS/2のデバイスドライバはリアルモード，プロテクトモードの両方で動作する必要がある．このためOS/2は，バイモーダルポインタなどの特殊なサービスを提供している．

デバイスモニタ

ある特定のキャラクタ型デバイスドライバのデータストリームを横取りして，これに追加・変更を加えるようなプログラムを作成し登録することができる．このようなプログラムをデバイスモニタと呼ぶ．

デバイスドライバの取り外し

OS/2のキャラクタ型デバイスドライバは，取り外しが可能である．OS/2で，以前ロードされたデバイスドライバと同じ論理名をもつデバイスドライバがロードされると，カーネルが以前のデバイスドライバに置き換えられてもいいかどうかを問い合わせ，よければ新しいデバイスドライバを導入する．以前のデバイスドライバは，それまで自分が所有していたすべてのリソースを解放し，再利用可能な状態にする．すでにロードされているデバイスドライバが取り外しを拒否すれば，カーネルは新しいデバイスドライバの導入を中止する．

OS/2デバイスドライバの種類

OS/2デバイスドライバには，MS-DOS同様キャラクタ型デバイスドライバとブロック型デバイスドライバがある．これらはまとめてカーネルデバ

イスドライバと呼ばれている．

PMには，これらのカーネルデバイスドライバ以外に表示ドライバと呼ばれるプログラムが必要になる．表示ドライバはダイナミックリンクライブラリであってデバイスドライバではないのだが，PM起動時の出力装置のサポートに必要である．

キャラクタ型デバイスドライバ

キャラクタ型デバイスドライバは，端末やプリンタなどのように1バイト単位で入出力処理を行うデバイスを制御する．

キャラクタ型デバイスドライバは，1つの実行形内に複数のデバイスドライバ含めることができる．それぞれのキャラクタ型デバイスドライバは，1～8文字の論理名をもち，この名前でファイルのようにオープンされる．

ブロック型デバイスドライバ

ブロック型デバイスドライバは，データを1バイト単位ではなくひとまとめにして転送する周辺装置を制御する．ブロック型デバイスは，ディスクドライブのようなランダムアクセスが可能なデバイスに指定される．

ブロック型デバイスドライバは，複数の物理ユニットをサポートでき，1つの物理ユニットに複数の論理ユニットを割り当てることも可能である．

ブロック型デバイスドライバは，論理名ではなく1文字のドライブ識別子によって区別される．ドライブ識別子は，ブロック型デバイスドライバがロードされた順番によって決められ，割り当てられる文字の総数は，そのドライバがサポートする論理ユニットの数によって決まる．

表示ドライバ

表示ドライバとは，特権レベルのリング2で動作する専用I/Oルーチンを含むダイナミックリンクライブラリである．表示ドライバは，表示装置への内部インターフェースとカーネルデバイスドライバおよび画面へのOS/2インターフェースとの間のリンクを提供する．

表示ドライバには，ディスプレイ表示ドライバ，プリンタ表示ドライバ，プロッタ表示ドライバなどがある．

アマチュアにはとても無理！ OS/2 PMのプリンタドライバを作る

イースト株式会社 第一開発部第一開発課 主任 **国分秀昭**

●学習の難しさ
表示ドライバはOS/2のどこに位置付けられるのか

OS/2のプレゼンテーションマネージャにおけるプリンタドライバとは，カーネルとデバイスの間のものではなく，プレゼンテーションマネージャとカーネルの間の表示ドライバのことです．

プリンタドライバを書くのにもっとも苦労したのは，表示ドライバがOS/2というシステム中でどういう位置にあって，どういうことをやらなければならないのかを理解することでした．実際，私もプリンタドライバの勉強をはじめてから，なにかプリンタに出てくるまでに5か月ほどかかりましたから（苦笑）．

以前，MS-DOSでRS-232Cのデバイスドライバを書いたことがありますけど，そのときの知識が役に立ったのは最初の2～3日だけでしたね．結局，OS/2のプリンタドライバを理解するのに5か月かかったわけです．

プログラミングマニュアルを読んでも大まかな説明しか書かれていないので，具体的にはよくわからないことが多いんです．こういうときは，サンプルやDISPLAY.DLLのソースをみて，具体的な位置付けなどを理解していくしかありませんでした．

MS-DOSのデバイスドライバは，ほんとうにハードウェアに依存したことをやっていますので，ハードウェア，とくにLSIなどの仕様を知っているというのが重要になるんですが，OS/2はそれ以上にシステムのことを知っている必要がありますね．

●表示ドライバの動作
ディスパッチテーブルで関数の先頭アドレスを上書き

表示ドライバがメモリ上にロードされるときに，ディスパッチテーブルというものを渡されます．このディスパッチテーブルがなにかといいますと，必要な機能，つまり関数の先頭アドレスの配列です．

OS/2のプリンタドライバは，そのディスパッチテーブルを渡されて，本来GpiのDLLがそのディスパッチテーブルを参照して処理を呼ぶわけなんですが，そこを自分のプリンタドライバ内部の関数の先頭アドレスで上書きすることによって横取りしてしまいます．

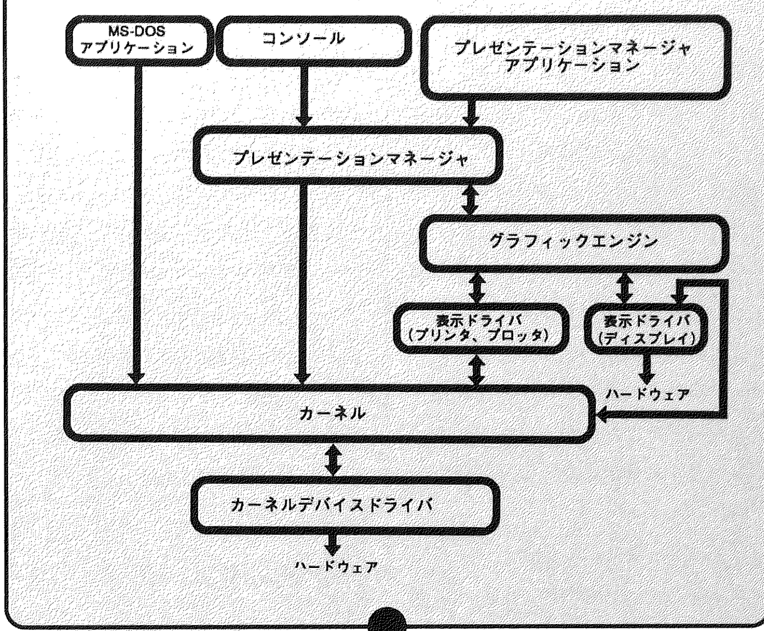
プリンタドライバが必ずサポートしなければいけない処理があるんですが，そこをプリンタドライバ内部の関数の先頭アドレスに上書きしてしまうわけです．すると，Gpiの処理をアプリケーションが呼んだときにグラフィックエンジンがそのディスパッチテーブルを参照して，どこを呼べばいいのかをみて，そ

OS/2デバイスドライバの構造

OS/2のデバイスドライバは，EXEヘッダ，デバイスドライバヘッダ，データ，コード，初期化ルーチンという構造をしている（図6）．

また，デバイスドライバを構成するルーチンとして，初期化ルーチン，ストラテジルーチン，ハードウェア割り込みルーチン，タイマ割り込みルーチン，ソフトウェア割り込みルーチン，モニタサポートルーチンの6つがある．

図5
OS/2システム中のデバイスドライバ／表示ドライバの位置



こを呼び出すわけです．

ディスパッチテーブルの元の関数の並びというのは，グラフィックエンジン内部の関数の先頭アドレスで，プリンタドライバが自分で処理できないものについてはグラフィックエンジンが処理しています．

プリンタ用の表示ドライバは，ディスプレイ用の表示ドライバとほとんど内容は同じで，違いはマウスのポインタを表示するかしないかという点くらいです．

実際，OS/2の DDDK に入っているプリンタドライバのソースは，おもな処理を DISPLAY.DLL を呼び出すことで行っています．これはどういうことかといいますと，グラフィックエンジンの行っている処理は相当複雑で，これをプリンタの機種ごとに実現するというのは，たいへん難しいですし，相当複雑な処理を要求されますので，その部分をすべて DISPLAY.DLL にまかせてしまっているわけです．

こうすることで，ディスプレイに表示しているものとプリンタから出力されるものが一致する WYSIWYG が，理想的な形で実現されるわけです．

ただし，こうなるとパソコンがもっているフォントしか使えませんので，印字された文字の品位はよくありません．そこで，私達が作成しているプリンタドライバでは，そのへんは DYSPLAY.DLL を使用しないで自前で処理しています．

●プログラミングの問題点
プログラマ全員がシステムを理解することが必要

プリンタドライバに必要とされている処理は，システムから規定されています．そこで，プログラマがなにをするかというと，プリンタの能力に合わせて規定されている処理を実現するわけです．

実際のプログラミングでもっとも時間がかかるのは，モジュール設計です．つまりディスパッチテーブルを上書きするときに，どれとどれをプリンタドライバで実現し，どれとどれをグラフィックエンジンにまかせるのか，という問題です．

アプリケーションを開発するときには，1つのプログラムをモジュールごとに分けて，複数のプログラマがプログラミングを行うと思います．その場合，1人の人間が OS/2のシステム全体の知識をもっているということはまずないでしょう．1人1人は，専門的な知識しかもっていないはずです．

しかし，プリンタドライバの場合は，各プログラマがシステム全体を頭に入れておかないと混乱してとてもじゃないけどできません．

たとえば，Gpi の関数の仕様がきちんと頭に入っていないと，グラフィックエンジンの関数をプリンタドライバで実現するときに，どうやったらいいのかわからなくなってしまいます．

先日，やはり OS/2のプリンタドライバを作成したんですが，ベータバージョンまでに16人月，最終的には20人月ほどかかりました．私どもが作っているデバイスドライバですと，実行ファイルがリソースを含めて300KB ほどあります．まぁ，ちょっと1人でやれる量ではありませんね．

デバイスドライバのプログラムモデル

OS/2のデバイスドライバは，コードセグメントとデータセグメントを複数もつことができる．ただし，最初のセグメントはデータセグメント，2番目のセグメントはコードセグメントでなければならない．OS/2は，この2つのセグメントをメモリの低い位置に固定し，他のセグメントをメモリの高い位置に置く．最初の2つのセグメントは，デバイスドライバ初期化後もシステム内に保持されるが，ほかのセグメントはとくに指定しない限り保持されない．

リアルモードでは，最初のコードセグメントとデータセグメントのみに直接アクセス可能なので，リアルモードで必要なコードおよびデータはこの部分に含まれている必要がある．

デバイスドライバヘッダ

デバイスドライバヘッダは，常にデバイスドライバのデータセグメントの最初にある．デバイスドライバヘッダには，そのデバイスドライバに関する情報が収められており，カーネルは入出力要求を処理する際にこれを調べる．

デバイスドライバヘッダの“名前”フィールドには，キャラクタ型デバイスドライバでは論理名が含まれ，ブロック型デバイスドライバでは最初の1バイトにドライバがサポートする論理ユニットの数が含まれる．

初期化ルーチン

デバイスドライバの初期化を行うルーチンで，デバイスヘルパルーチンのエントリポイントの保持と，デバイスドライバの最終オフセットの設定を行う．

OS/2が起動すると，CONFIG.SYSファイルに書かれた順にデバイスドライバが組み込まれる．カーネルは，リンク順にデバイスドライバに初期化要求のリクエストパケット（デバイスヘルパルーチンのエントリポイントが格納されている）を送る．

初期化が終了すると，初期化ルーチンは最終オフセットを設定して初期化ルーチン自体をディスカード可能にする．ディスカードされた領域は，カーネルによって空きメモリとして利用される．

ストラテジルーチン

ストラテジルーチンは，デバイスドライバのエントリポイントである．アプリケーションが出す入出力要求を受けて，カーネルはデバイスドライバのストラテジルーチンにリクエストパケットを渡す．ストラテジルーチンは，リクエストパケット内のコマンドコードによりデータセグメントのディスパッチテーブルを利用して各処理ルーチンに分岐し，処理を実行する．処理が終了するとリクエストパケットにステータスを設定し，カーネルへリターンする．

ハードウェア割り込みルーチン

ハードウェア割り込みルーチンは，デバイスからの割り込みによって入出力処理を行う．

入出力処理はCPU時間を多く使用するため，ストラテジルーチンのみで処理するとシステム全体の稼働効率が低下する恐れがある．このため，ストラテジルーチンは入出力要求をリクエストキューに入れ，すぐにカーネルに

図6
デバイスドライバの構造

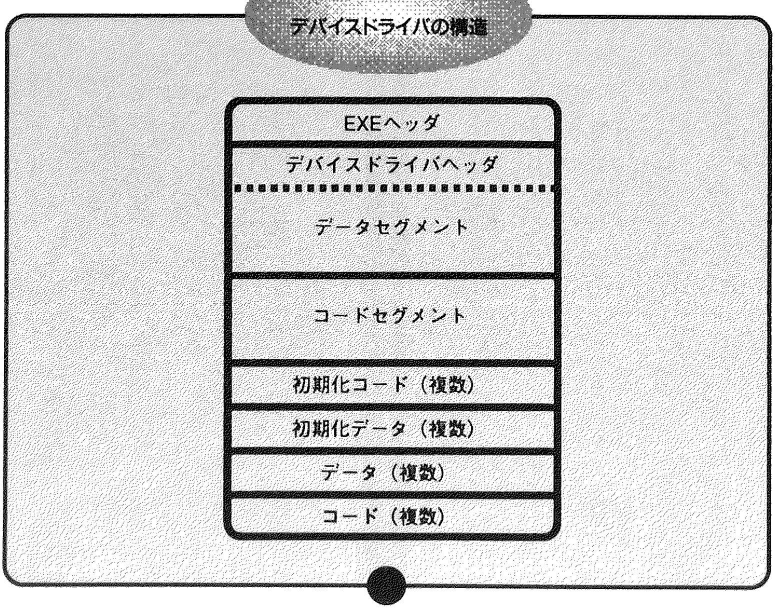


図7
デバイスドライバヘッダ

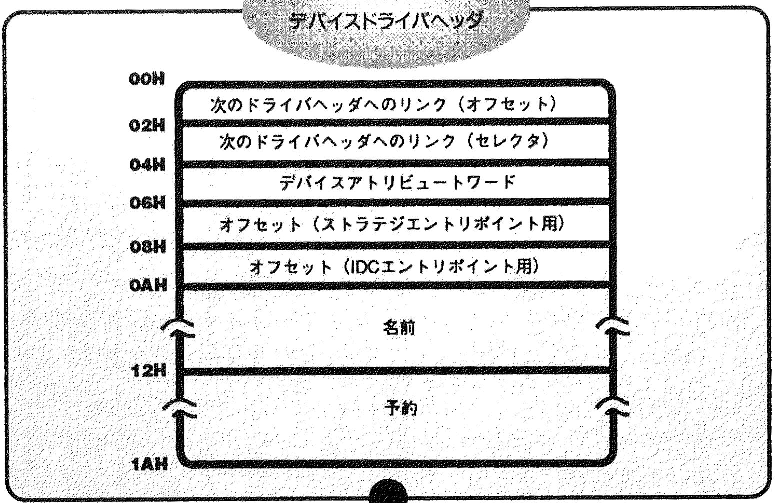
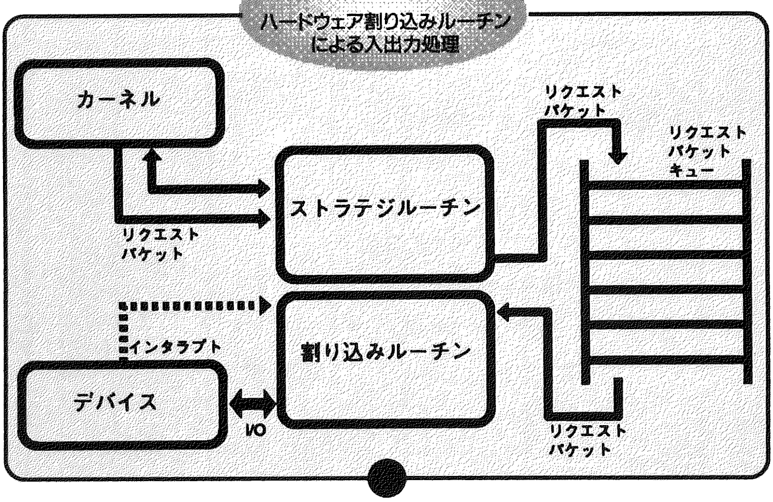


図8
ハードウェア割り込みルーチンによる入出力処理



リターンし，実際の入出力処理をハードウェア割り込みルーチンにまかせることが多い．

タイマ割り込みルーチン

これは，ハードウェア割り込みを発生しないデバイスとの入出力やタイムアウトの発行で利用される．

ハードウェア割り込みのかわりにタイムチェックを利用してタイマ割り込みルーチンが起動し，入出力を処理する．

ソフトウェア割り込みルーチン

DOS互換モードで動作するMS-DOSアプリケーションのINT 21HによるソフトウェアI割り込みは，OS/2内部でエミュレートされ，カーネルに渡される．しかし，OS/2ではINT 21H以外のソフトウェア割り込みは受け付けない．

OS/2のデバイスドライバは，INT 21H以外のソフトウェア割り込みを受け取るためにデバイスヘルパルーチンを利用してソフトウェア割り込みのベクタテーブルを書き換えることができる．こうすると，ソフトウェア割り込みによる処理要求はカーネルを通さず直接デバイスドライバに渡されるようになる．

モニタサポートルーチン

このルーチンを利用してデバイスモニタを実現する．モニタサポートルーチンはモニタバッファを作成し，モニタバッファに入出力を行う．

OS/2デバイスドライバの動作

オペレーションモード

OS/2のデバイスドライバは，初期化モード，カーネルモード，割り込みモード，ユーザーモードのいずれかで動作する．このモードによって，デバイスドライバが使用できるデバイスヘルパルーチンと実際の動作が変わってくる．

初期化モード

OS/2が，起動時にデバイスドライバをロードし，これを初期化するためにデバイスドライバをコールするときのモード．このときデバイスドライバは，アプリケーションのように特権レベルのリング3で動作する．したがっ

て，このモードでは，OS/2のAPIの利用が可能である．

●●● カーネルモード

通常のアプリケーションによるAPIのコールによって入出力処理を開始するためにデバイスドライバのストラテジエントリポイントがコールされるとき，デバイスドライバはカーネルモードにある．このときデバイスドライバはリング0で動作し，ほとんどのデバイスヘルパルーチンを利用できる．

●●● 割り込みモード

ハードウェア割り込みによってデバイスドライバが実行されるとき，デバイスドライバは割り込みモードで動作する．カーネルモードと同様，デバイスドライバはリング0で動作するが，デバイスヘルパルーチンの一部しか利用できない．

●●● ユーザーモード

ユーザーモードは，DOS互換モードで実行中のアプリケーションからソフトウェア割り込みによって入出力処理が要求されたときのモードで，このモードのデバイスドライバは他のモード時と異なりプリエンプト可能である．

リクエストパケット

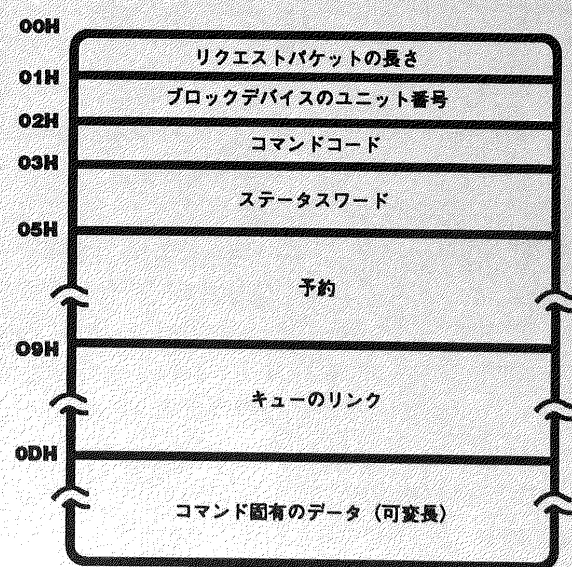
リクエストパケットは，カーネルとデバイスドライバの間の通信に利用されるデータ構造体で，要求された処理やデバイスドライバの状態，リクエストキューのリンク情報などを含む13バイトのヘッダとデバイスドライバに行わせる処理ごとの情報を収めたコマンド指定データからなる（図9）．

リクエストパケットのアドレスは，リアルモードでもプロテクトモードでも同じアドレスを指すようにバイモーダル形式で渡される．

基本的な制御フロー

1. アプリケーション中のスレッドが，デバイスに対してDosOpenというAPIによるコールを行う．
2. カーネルはDeviceOpenというリクエストパケットを作成し，デバイスドライバのストラテジルーチンに渡す．
3. ストラテジルーチンは，リクエストパケットをチェックして必要な処理を行うルーチンに分岐する．
4. アプリケーションのスレッドは，返ってきたデバイスハンドルを用いてDosRead，DosWriteを発行して入出力を行う．
5. カーネルは，デバイスドライバにリクエストパケットを送ってINPUT，OUTPUT命令を出し，必要ならスレッドを停止し他のスレッドにCPUを明け渡す．
6. 割り込みルーチンが入出力処理を行い，終了するとデバイスヘルパルーチンを利用してカーネルにその旨を通知する．
7. カーネルはスレッドを再開し，スレッドはDosCloseを発行する．
8. カーネルはデバイスドライバに対してDeviceCloseを発行し，デバイスドライバは事後処理を行う．

図9
リクエストパケットの構造



表示ドライバの機能

表示ドライバは，PMおよびそのアプリケーションからI/O要求を受け取り，その要求を装置に固有な属性によって処理する専用I/Oルーチンである．表示ドライバはダイナミックリンクライブラリであり，これは特権レベルのリング2で実行される．

インターフェース

表示ドライバのインターフェースは，ディスパッチテーブル内のポインタを介して呼び出されるひと組みのグラフィックエンジン機能から構成されている．つまり，ディスパッチテーブルは，機能処理ルーチンに対するポインタの配列である．

PMが初期設定されると，ディスプレイ表示ドライバが自動的にロードされる．他の表示ドライバは，アプリケーションがDevOpenDCを発行したときにロードされる．

ロードされた表示ドライバにはデフォルトディスパッチテーブルのコピーが与えられ，ドライバ内のenableルーチンはこのコピーを修正し，ドライバがサポートしている機能処理ルーチンをポインタによってアドレスできるようにする．

ディスパッチテーブルには，表示ドライバがサポートしなければならない必須機能とグラフィックエンジンによってサポートされる機能の2つがリストされている．

デバイスコンテキスト

PM上のグラフィックスは，すべてGpi(Graphics Programming Interface)によって実現されている．Gpiライブラリを利用するには，まずプレゼンテーションスペースを定義する必要がある．プレゼンテーションスペースとは，グラフィックスデータをデバイスコンテキストに置き換えるために必要な情報をもつ内部的なデータ構造体である．デバイスコンテキストもやはりデータ構造体で，ディスプレイ，プリンタ，プロッタなどの物理デバイスを定義している．

アプリケーションは，DevOpenDCを使用してデバイスコンテキストをオープンし，これにプレゼンテーションスペースを関連付け，そのスペースに書き込みまたは描画を行う．DevOpenDCが発行されるたびに，デバイスコンテキストのインスタンスが1つ作成される．このインスタンスは内部的にディスパッチテーブルとみなされ，アプリケーションはこのディスパッチテーブルを介してそのデバイスコンテキスト用の表示ドライバ内の処理ルーチンを呼び出す．

エクスポート／インポート

表示ドライバは，必要な機能を実現するためにいくつかのエントリポイントをエクスポートし，また，内部エンジン機能にアクセスするためにグラフィックエンジンからInnerGreEntry機能をインポートする必要がある．

必須機能

すべての表示ドライバがサポートしなければならない機能を次に示す．

- 線機能
- 走査機能
- ビットマップ機能
- ストリング機能
- マーカー機能
- 属性機能
- 装置機能
- カラーテーブル機能
- エスケープ機能
- その他の装置機能

シミュレートされた機能

グラフィックエンジンシミュレーションルーチンによって実現される機能を次に示す．

- 弧機能
- パス機能
- 領域機能
- クリップ機能
- 変換機能

OS/2デバイスドライバの作成について

実際にデバイスドライバまたは表示ドライバを作成するには，OS/2を発売しているハードウェアメーカーからプログラマーズツールキットまたはDDDK (Device Driver Development Kit) などと呼ばれている専用開発キットを入手する必要がある．これらの入手方法については，メーカーに直接問い合わせていただきたい．

(編集部)

［協力］
イースト株式会社

PRODUCTS REVIEW

MS-C 6.0に続くプロ指向の言語パッケージ

Microsoft BASIC Professional Development System Ver.7.1

海老原浩之

マイクロソフト㈱から，プロフェッショナル向けのBASIC開発言語として，「Microsoft BASIC Professional Development System Ver.7.1」が発売される（以下MS-BASIC Ver.7.1と略す）．MS-BASIC Ver.7.1は，米国ではすでに昨年から発売されているものであり，日本においてはMS-BASIC Ver.5.3以来，ひさびさのニューバージョンということになる（その間に初心者向けにQuick-BASIC Ver.4.2，Ver.4.5が発売されている）．

同社がProfessional Development Systemと命名してる開発言語には，MS-C Ver.6.0がすでにあるが，今回それについで2つ目のプロフェッショナル向け言語が発売されることになる．では，この新しく発売されるMS-BASIC Ver.7.1について述べていくことにしよう．

特徴

この新しいMS-BASIC Ver.7.1は，以下のような5つの特徴をもっている．

- プロジェクト指向
- EMSをサポートした大規模プログラムの作成
- ビジネスプログラミング支援
- MS OS/2のサポート
- 強力なオプティマイズ機能

では，この5つの特徴について説明する．

プロジェクト指向

プロジェクト指向というのは，すでに同社の製品であるMS-C Ver.6.0のPWB（Programmer's Work Bench）やQuickC，QuickBASICに取り入れられているコンセプトであり，それが，MS-BASIC Ver.7.1にも採用されたわけである．これによって，エディット，コンパイル，リンク，実行，デバッグの一連の作業を管理する統合環境が実現した．

MS-BASIC Ver.7.1には，QuickBASIC Ver.4.5よりも優れたQuickBASIC Extended Versionが付属している．このため，いままでQuickBASIC Ver.4.5を使って開発を行ってきた人でも違和感なく使用することができるだろう．

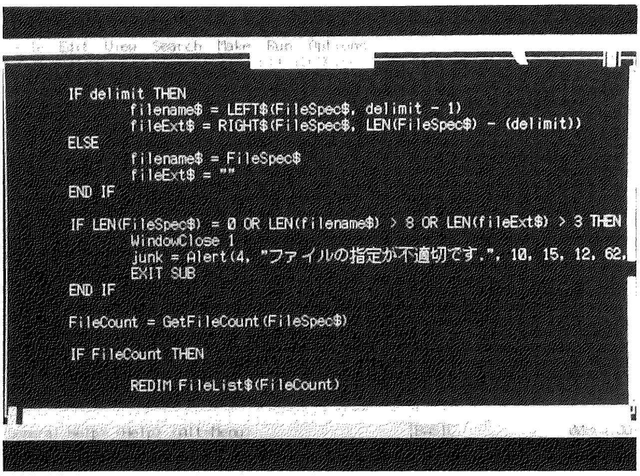
さらに，ほかの開発言語と組み合わせたアプリケーション開発やOS/2用のアプリケーション開発も，PWBによる統合環境下で行うことが可能になっている．

EMSを利用した大規模プログラムの作成

EMSを利用することによって，大規

NEC PC-98シリーズ一覧
PC-9801DX2/U2（1990/11発表）●CPU:80286（V30（8MHz）相当/10/12MHz）●メインメモリ：640KB●スロット数：4●備考：98RX/EX後継機，FM音源

写真1
PWB



模なプログラムとデータ領域を確保することが可能になった. 付属のQuick-BASIC Extended Versionにおいては, 512バイトから16KBまでのプロシージャと数値配列をEMSに展開したり, QuickBASIC Extended Version本体を約60KB分HMA (High Memory Area) に移動したりして, プログラム用の領域を拡大することができる. EMSやHMAが導入済みの場合, プロシージャは自動的にEMSに展開し, 数値配列は/Eaオプションをつけて起動することでEMSに展開する. さらに, QuickBASIC Extended Version本体も自動的にHMAに移動する.

また, EMSとディスクオーバーレイを採用したことで, 640KBの制限を超えて最大16MBまでのプログラムを作成することが可能になった.

ビジネスプログラミング支援

ビジネスアプリケーションを作成するうえで, データの入出力やデータファイルの扱いは重要な位置を占めるものである. それらを補助するツールや命令などが, 今回のバージョンからサポートされた.

・索引ファイル (ISAM FILE) のサポート

・ツールボックスライブラリ

の2つである.

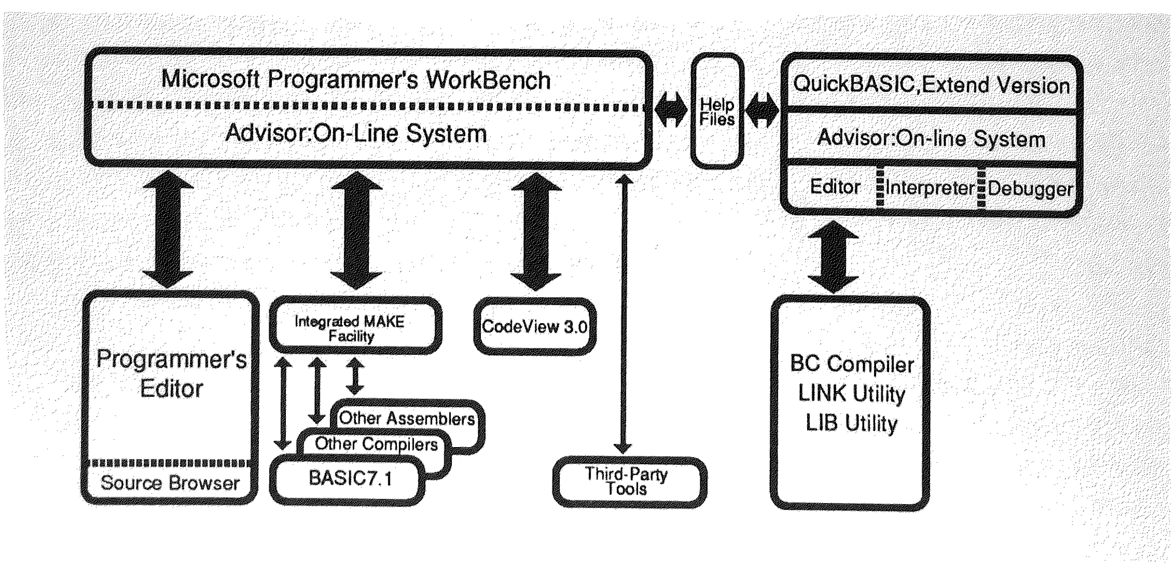
■索引ファイル (ISAM FILE)

索引ファイルは, COBOL言語ではあたりまえに使えるファイルアクセスの方法だが, いままでのBASIC言語ではできなかった. 無理に行おうとすれば, 自前で索引ファイルのロジックを作成しなければならなかった.

しかし, MS-BASIC Ver. 7. 1では標準で索引ファイルが扱えるようになり, BASICにおいても高速なデータ検索やデータ操作が可能になった. これによって, BASICによるビジネスアプリケーション開発が実に容易になったといえる.

この索引ファイルを扱うロジックは, DOSの場合TSR (常駐型) とライブラリの2種類のタイプがあるので, プログラマはどちらか片方を自由に選択できる. OS/2においては, DLLによってサポートされている.

また, 最大128MBまでのデータを扱うことができ, ISAMのインデックスオーダーはMicrosoft Excelの50音ソーティングオーダーに準拠してい



る．さらに，ISAMデータをメンテナンスするユーティリティとして，以下の3つが提供されている．

・ISAMREPR. EXE
ISAMファイルの復旧ユーティリティ
・ISAMIO. EXE
TEXTファイル←→ISAMファイル双方向変換ユーティリティ
・ISAMPACK. EXE
ISAMファイル圧縮ユーティリティ

■ツールボックスライブラリ

ツールボックスライブラリとは，ユーザーインターフェース，グラフ表現，行列演算の3つが含まれたライブラリである．このライブラリには，ソースプログラムも付属するので，自分でさらにカスタマイズを行うことができるようになっている．各ツールボックスは以下のような機能をもっている．

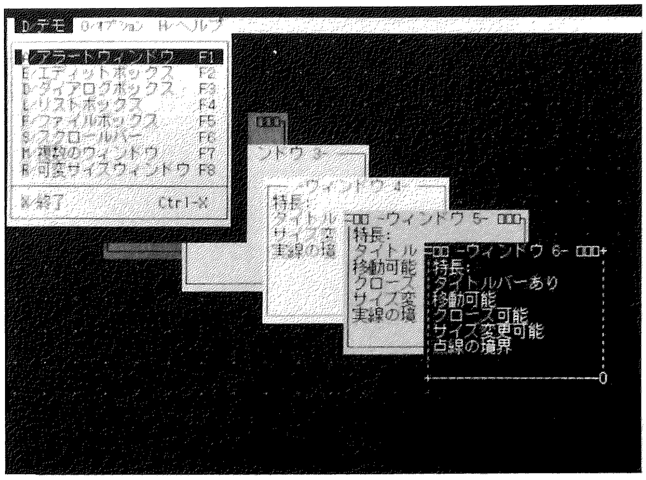


写真2 ユーザーインターフェースツールボックス

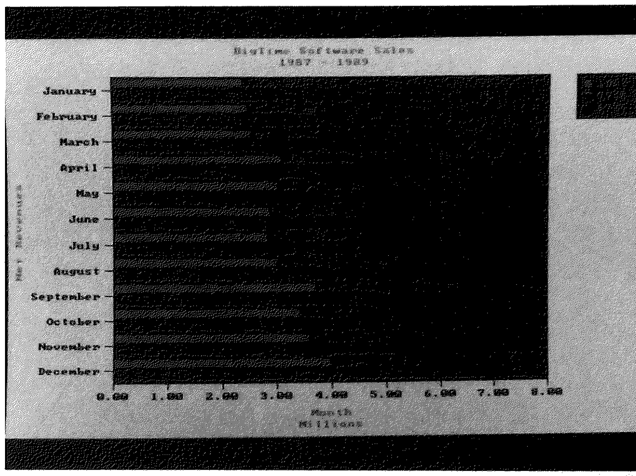


写真3 プレゼンテーショングラフィックスツールボックス

ユーザーインターフェースツールボックス

Microsoft WORKSやQuick-BASIC, QuickCのようなキャラクタ型ウィンドウインターフェースを簡単に構築できる（写真2）．

たとえば，ドロップダウンメニューやダイアログボックス，マルチウィンドウ表示などをアプリケーションプログラムに簡単に付加することができるわけである．

プレゼンテーショングラフィックスツールボックス（グラフ表現）

円グラフ／縦棒グラフ／横棒グラフ／線グラフ／散布図などを簡単に描くことができ，MS－Windowsのビットマップフォントを利用した文字をグラフィック画面に表示できるようになっている（写真3）．

行列演算ツールボックス

よく使う行列演算が，BASICのルーチンとして含まれている．たとえば，行列の加減乗除，ガウスの消去法を使った連立方程式の解法，行列式や逆行列の計算などがサポートされている．

MS OS/2のサポート

MS-DOS上のみでなく，MS OS/2上でもBASICが扱えるようになった．これによって，いままでは，MS-Cでしか行えなかったOS/2用のアプリケーション開発が，ポピュラーな開発言語であるBASICで行えるようになったわけである．まだまだ世の中には，BASICで書かれたアプリケーションがたくさんあると思う．それらのアプリケーションをDOSばかりでなくOS/2でも動作させることが可能になったわけである．

さらに，DOS上で開発したMS-BASIC Ver. 7.1のプログラムは，基本的には何も変更せずにOS/2上で動作するので，C言語と比べて比較的簡単にOS/2のアプリケーションが作成できることになる．

強力なオプティマイズ機能

いままでのBASICコンパイラでは，できあがった実行モジュール（.EXE）が非常に大きなものになっていた．こ

れは，使われていない命令に関するモジュールまでリンクしていたためだが，MS-BASIC Ver.7.1では，MS-Cのオプティマイズ技法を取り入れたことにより，コンパクトで高速な実行モジュールが作成されるようになった．

とくに，グラフィックス系を中心に使われていない命令に関するオブジェクトモジュールをリンクしないようにしたり，IF論理式の評価の省略(図1)，共通部分式の利用（図2），PEEP-HOLINGの増加，ジャンプの最適化などにより，同じソースプログラムをコンパイルしても，従来のQuick-BASICと比較して約30%～80%も小さい実行モジュールを作成することができるようになった．実際に画面に時計を表示させるプログラムをQuick-BASIC Ver.4.5とMS-BASIC Ver.7.1でコンパイルしたところ，MS-BASIC Ver.7.1のほうがはるかに小さくなった（図3）．

図1
IF論理式の評価の省略

```
ソースプログラム
IF (A%>0) AND (B%>0) THEN PRINT A%
論理式評価の省略が行われないときのコンパイル結果
        cmp     word ptr A%,00h
        mov     ax,0000h
        jng     S+01h
        dec     ax
        cmp     word ptr B%,00h
        mov     cx,0000h
        jng     S+01h
        dec     cx
        and     cx,ax
        and     cx,cx
        jne     S+03h
        jmp     100001
        push    A%
        call    BSPE12
100001:

論理式評価の省略が行われたときのコンパイル結果
        cmp     word ptr A%,00h
        jng     100001
        cmp     word ptr B%,00h
        jng     100001
        push    A%
        call    BSPE12
100001:
```

図2
共通部分式の利用

```
ソースプログラム
A%=B%+C%
D%=B%+C%+E%
共通部分式を利用してない場合のコンパイル結果
        mov     ax,C%
        add     ax,B%
        mov     A%,ax
        mov     ax,C%
        add     ax,B%
        add     ax,E%
        mov     D%,ax

共通部分式を利用した場合のコンパイル結果
        mov     ax,C%
        add     ax,B%
        mov     A%,ax
        add     ax,E%
        mov     D%,ax
```

図3
実行形の大きさの比較

オリジナルソース	TOKEI.BAS	2488バイト
QuickBASIC Ver.4.5	TOKEI.EXE	40726バイト
MS-BASIC Ver.7.1	TOKEI.EXE	12782バイト

QuickBASICに比べて約32%ほどの大きさですんでいる．
MS-BASIC Ver.7.1は，ベータ版リリース2での結果である．

QuickBASIC Extended Version

MS-BASIC Ver.7.1には，QuickBASIC Extended Versionが付属している．DOS環境においてプログラムを開発するなら，PWB環境を使うよりQuickBASIC環境のほうがよい．ただし，DOS環境においても複数の言語を組み合わせて開発する場合は，PWBのほうが優れている．

QuickBASIC Extended Versionは，QuickBASIC Ver.4.5よりもさまざまな点において機能強化がなされている．たとえば，カスタマイズ可能なUtility MENU，コンパイラ制御の改善，ヒストリカルUndo／Redoコマンド，プリント出力先の変更，エディタのキーアサイン変更，デバッグ機能の強化などである．

また，先に「EMSを利用した大規模プログラムの作成」のところでも説明したが，EMSやHMAを利用できるようになっている．そのほかに，far文字列のサポートにより，文字列を最大64KBまで取れるようにしたり，/NO-FRILLSスイッチによってUtility/Help/Optionの機能を停止させて，約19KBのメモリをユーザーメモリとして解放したりしている．

このQuickBASIC Extended Versionがあれば，従来のQuick-BASIC Ver.4.5は必要ないといっていいだろう．

QuickBASIC Ver.4.5とMS-BASIC Ver.7.1の違い

ユーザー定義型変数中の静的配列宣言

ユーザー定義型の変数中に，静的配

列を使えるようになった．これによって，複雑なデータをBASICで扱うことができるようになり，ISAMとこの機能を利用することで複雑な構造をもつデータベースを簡単に構築することが可能になった．

ローカルエラートラップ機能

今までは，モジュールでしかエラートラップ処理ができなかったが，プロシージャ内でもエラートラップを処理することが可能になった．

ファイル操作関数／ステートメントの追加

従来はなかったファイル操作関連の関数が，MS-BASIC Ver. 7. 1では追加されている．追加されたものは，DIR$（ディレクトリ内のファイル名を返す），CURDIR$(カレントパスを返す)，CHDRIVE(指定したドライブに変更する)，CHDIR（指定したディレクトリに変更する）などの関数，ステーメントである．これによって，MS-Cにより近いファイル操作が行えるようになった．

エラーレベルの設定

END, STOP, SYSTEMなどでプログラムを終了したとき，MS-DOSやOS/2へエラーレベルを返せるようになった．これによって，バッチファイル内での分岐処理が実現された．

スタックサイズの変更

プログラム実行中に，スタックサイズを変更することができる．

PRESERVEオプションの追加

REDIMステーメントにPRESERVEオプションを指定することで，配列のサイズを変更するときに以前のデータを保護することができるようになった．これによって，配列に必要なメモリを制御できる．

ISAM操作関数／ステーメントの追加

先に説明した，索引ファイル(ISAM)を制御する関数，ステーメントが追加された．

開発支援ツールの充実

MS-BASIC Ver. 7. 1には，MS-C Ver. 6. 0と同様の開発支援ツールが付属している．PWB，NMAKE，CodeView 3. 0，オンラインヘルプなどである．

すでにMS-C Ver. 6. 0を使っている人は，MS-BASIC Ver. 7. 1のユーティリティを使ったほうがよいと思う．なぜなら，ユーティリティ類はMS-BASIC Ver. 7. 1もMS-C Ver. 6. 0も同じものだが，MS-BASICVer. 7. 1に付属しているもののほうが一部バージョンが新しくなっているからである．実際，PWBの速度がMS-C Ver. 6. 0に付属しているものよりも上がっていた．

そのほかの注意点など

QuickBASIC Ver. 4. 5で作成したライブラリは，そのままではMS-BASIC Ver. 7. 1では使えないので，再コンパイルしてライブラリを再作成する必要がある．

グラフィックス関連は，QuickBASIC Ver. 4. 5とMS-BASIC Ver. 7. 1では異なるので注意してほしい．MS-BASIC Ver. 7. 1は，デフォルトでテキストモードとなっているが，QuickBASIC Ver. 4. 5はデフォルトでグラフィックモードである．このため，SCREEN命令など一部変更しなければならないところがある．

環境変数の設定に関して，MS-C Ver. 6. 0とMS-BASIC Ver. 7. 1ではPWBを使う場合にかぎりINIT変数に注意してほしい．この中に作成されているTOOLS. INIとCURRENT. STSがMS-C Ver. 6. 0のまま残っていると，MS-BASIC Ver. 7. 1のモードでPWBを起動した場合，TOOLS. INIとCURRENT. STSはMS-BASIC Ver. 7. 1用に書き換えられてしまう．

PWBを使う環境においては，MS-C Ver. 6. 0とかMS-BASIC Ver. 7. 1に分けることはせず，PWB用のディレクトリを作成して，MS-C Ver. 6. 0とMS-BASIC Ver. 7. 1の両方から参照できるようにしておくのが便利であり，HDD容量の無駄使いをせずにすむ．

MS-BASIC Ver. 7. 1では，実行モジュール(. EXE)のサイズを小さくすることに力が注がれたようだ．しかし，実行モジュールが小さくなったからといって，グラフィック表示の速度がアップしているわけではないので注意してほしい．基本的には，QuickBASIC Ver. 4. 5と同じである．この点は，次のバージョンアップで対応してほしいものである．

MS-C Ver. 6. 0とMS-BASIC Ver. 7. 1を比べた場合，どちらが優れているか現時点で判定するのは難しいところだ．プログラムによって，MS-C Ver. 6. 0が有利であったり，MS-BASIC Ver. 7. 1のほうが有利であったり，あるいはMS-C Ver. 6. 0とMS-BASIC Ver. 7. 1が同等の場合などがある．

ただし，QuickBASIC Ver. 4. 5は入門者用，MS-BASIC Ver. 7. 1はプ

ロフェッショナル向きということだけはいえる.

MS-BASIC Ver.7.1のスペック一覧

MS-BASIC Ver.7.1は,プロフェッショナル向きということもあるのだろうが,インストールするとHDDを13MBも占めるほどの大きさがある.もう,BASICだからといってバカにできるようなしろものではない.

MS-BASIC Ver.7.1のスペック一覧を表1にまとめてみたので参考にして欲しい.

最後に

MS-BASIC Ver.7.1について簡単に紹介してみた.説明不足のところもあったと思うが,多少は理解していただけたと思う.世の中には,まだまだBASICが使いやすいと思ってる人も多いことだろう.その人たちにとっては,非常に魅力ある製品が出てきたと思う.

実際,生成される実行モジュールが小さくなるのは非常に喜ばしいことだし,OS/2においてもBASICで開発ができるというのは実にありがたい.しかし,PWBに関してはMS-C Ver.6.0のものより多少速度の改善が見られたが,まだまだ遅い.今後の改良に期待したいものである.

表1　MS-BASIC Ver.7.1のスペック一覧

●変数の型

整数	INTEGER	-32,768 to 32,767
長整数	LONG	-2,147,483,648 to 2,147,483,647
単精度実数	SINGLE	絶対値1.40129E-45 to 8.403823E+38
倍精度実数	DOUBLE	絶対値4.94065D-324 to 1.79769313486231D308
通貨	CURRENCY	-922337203685477.5808 to 922337203685477.5807
可変長文字列	STRING	0 to 32,767バイト
固定長文字列	STRING*n	0 to 32,767バイト

●QuickBASIC Extended Versionのオプションスイッチ(一部)

/AH	64KBを超える数値配列の利用
/C:buffersize	通信受信バッファサイズの変更
/Ea	512〜16KBの数値配列をEMSに移動
/E:n	EMSメモリの利用量の指定
/L libraryname	Quick Libraryの読み込み
/Nof	QBXの一部機能を解放,19KBのメモリを解放

●BCコンパイラのオプションスイッチ(一部)

/A	アセンブリファイルの生成
/Ah	64KBを超える数値配列の利用
/C:buffersize	通信受信バッファサイズの変更
/D	デバッグスイッチ
/FBr	PWB-Browser情報の生成
/FBx	PWB-Browser詳細情報の生成
/FPa	代替数値演算ライブラリの利用
/FPi	エミュレータライブラリの利用
/Fs	farストリングの利用
/Lp	MS OS/2用EXEファイルの作成
/Lr	MS-DOS用EXEファイルの作成
/O	独立型EXEファイルの作成
/Ot	スタックフレーム最適化
/Zd	SYMDEB用EXE作成
/Zi	CodeView3.0用EXE作成

●数値演算方式

内部演算は,IEEE規格に準拠.QuickBASIC Extended Versionでの浮動小数点演算は,エミュレートライブラリを利用.ユーザープログラムは,コンパイル時に代替数値演算ライブラリとエミュレータライブラリを選択可能.エミュレータライブラリは,数値演算コプロセッサを装備した機種において,非常に高速かつ高精度な演算を保証

●関数とステートメント

Microsoft BASIC 7.1は,QuickBASIC 4.5と高い互換性を保ちながら,ファイル管理やISAM操作,通貨型データタイプ,far文字列などに対応するために多数の関数,ステートメントを追加した.

●開発に必要なシステム

ハードウェア:	80286/386をCPUにもつPC-9800シリーズ,または,PC-98シリーズ,PC-H98シリーズ(ただし,ハイレゾリューションモードを除く)
OS:	MS-DOS 3.1以上もしくは,MS OS/2 1.1A以上
必要メモリ:	640KB(MS-DOS) 空きメモリ1MB以上(MS OS/2)
必要ディスク装置:	1MB容量のフロッピーディスク装置とハードディスク装置 ハードディスクの空き容量は,フルインストール時15MBが必要
推奨システム:	1MB以上のLIM4.0対応のEMSメモリ(MS-DOS),1MB以上のキャッシュディスク(MS-DOS) マウス
利用可能な日本語入力システム:	MS-KANJI API仕様の日本語入力システム (ATOK6,ATOK7の場合,付属ドライバで対応可能)

●作成された実行ファイルのターゲットシステム

ハードウェア:	PC-9800シリーズ,またはPC-98シリーズ,PC-H98シリーズ (ただし,ハイレゾリューションモードを除く)
OS:	MS-DOS 3.1以上もしくは,MS OS/2 1.1A以上
必要メモリ:	システムと作成プログラムに依存

●パッケージ内容

ディスク:	6枚
マニュアル:	3冊 Install Guide Programming Guide Reference

ユーザー登録カードなど

●価格	98,000円(税別)
●優待販売価格	Microsoft BASIC 5.3, Microsoft BASIC Compiler 5.2ユーザー 41,200円(税込) QuickBASIC 4.2, QuickBASIC 4.5ユーザー 72,100円(税込)
●発売時期	1991年6月末(予定)

プログラム遊工房

谷口典生

七乃巻
「三子棋（サン・ツー・チ）」ゲーム





今回は，思考型ゲームに挑戦してみました．三目並べの一種で単純なゲームですが，コンピュータが人間の相手をして打ってくれます．ちゃんと先読みをして打ってくるので，なかなか手ごわい相手です．
ワンポイントテクニックでは，基本的な思考ルーチンの構造について解説してみました．もっと複雑なゲームへ応用してみるとおもしろいと思います．

リスト1

```
/*
 * santuti.c              三子棋(サン・ツー・チ)
 */

#include <stdlib.h>
#ifdef __TURBOC__
/* Turbo C 用グラフィック初期化/復帰ルーチン */
#include <graphics.h>
#define fcircle(x,y,r)  fillellipse(x,y,r,r);
void graphinit()                          /* グラフィックモード設定 */
{
        int graphdriver, graphmode;

        graphdriver = DETECT;             /* 自動検出 */
        graphmode = PC98C8;               /* 8色モード */
        initgraph(&graphdriver, &graphmode, "");
}
void graphdeinit()                        /* グラフィックモード復帰 */
{
        restorecrtmode();                 /* 画面モードを元に戻す */
        printf("¥x1b*");                  /* 画面クリア */
}
#else
/* MS-C, Quick C 用グラフィック初期化/復帰ルーチン */
#include <graph.h>
#define BLACK                   0
#define BLUE                    1
#define RED                     4
#define WHITE                   7
#define setcolor                _setcolor
#define rectangle(l,t,r,b)      _rectangle(_GBORDER,l,t,r,b)
#define setfillstyle(a,b)       _setcolor(b)
#define fcircle(x,y,r) _ellipse(_GFILLINTERIOR,(x)-(r),(y)-(r),(x)+(r),(y)+(r))
void graphinit()                          /* グラフィックモード設定 */
{
        _setvideomode(_98RESSCOLOR);      /* グラフィック8色+テキストモード */
        printf("¥x1b[>5h");               /* カーソル消去 */
}
void graphdeinit()                        /* グラフィックモード復帰 */
{
        _setvideomode(_DEFAULTMODE);      /* デフォルト(テキスト80行)モード */
        printf("¥x1b[>5l¥x1b[>1l");       /* カーソル表示、最下行復帰 */
}
#endif

#define BSIZE           50                /* 1マスのサイズ */
#define BASE_X          30                /* 盤の位置(X) */
#define BASE_Y          20                /* 盤の位置(Y) */
#define PRADIUS         20                /* 駒の半径 */
#define CLEAR           0
#define MARK            2
#define n2x(n)          ((n)/3)           /* 盤上の位置 -> X座標 */
#define n2y(n)          ((n)%3)           /* 盤上の位置 -> Y座標 */
int board[9] = {0, 0, 0, 0, 0, 0, 0, 0, 0};     /* 盤データ */
int prev[5] = {0, 0, 0, 0, 0};                  /* 千日手判定用データ */
const int ctable[4]={RED, BLACK, WHITE, BLUE};

void showboard()                          /* 盤の表示 */
{
        int n;

        setcolor(WHITE);
        for(n=0; n<9; n++)
                rectangle(BASE_X+n2x(n)*BSIZE, BASE_Y+n2y(n)*BSIZE,
                        BASE_X+(n2x(n)+1)*BSIZE, BASE_Y+(n2y(n)+1)*BSIZE);
}

int bd2num(int *bd)              /* 盤データを数値化する(千日手判定用) */
{
        int n, val;

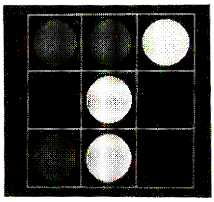
        for(n=0, val=0; n<9; n++)
                val = val*3 + bd[n] + 1;
        return val;
}

void put(int n, int pl)                   /* 駒の表示 */
{
        setfillstyle(SOLID_FILL, ctable[pl+1]);
        fcircle(BASE_X+n2x(n)*BSIZE+BSIZE/2, BASE_Y+n2y(n)*BSIZE+BSIZE/2,
                PRADIUS);
}

void mark(int n, int mode)                /* マーカーの表示/消去 */
{
        setcolor(mode ? RED : BLACK);
        rectangle(BASE_X+n2x(n)*BSIZE+2, BASE_Y+n2y(n)*BSIZE+2,
                BASE_X+(n2x(n)+1)*BSIZE-2, BASE_Y+(n2y(n)+1)*BSIZE-2);
}

void select(int *n)                       /* 場所を選択 */
{
        int inkey;

        do {
```



プログラムの使い方

このプログラムはPC-9801シリーズ用で，Turbo C, QuickCおよびMS-Cに対応しています．リスト1を打ち込み，コンパイルして実行ファイルを作ってください．

今回のプログラムではグラフィック関数を使っているので，MS-CやQuickCでコンパイルするときにはグラフィックライブラリを指定します．MS-Cなら「CL santuti.c /link graphics.lib」などとします．Turbo Cでは，実行時に.BGIファイルが必要となるので注意してください．

プログラムの実行は，そのまま

SANTUTI

としてください．

ゲームのルールと遊び方

三子棋は，中国で古くから行われている三目並べです．

盤は3×3のマス目で，白，赤3個ずつの駒を使用します．自分の駒を

```
 96:                 mark(*n, 1);                    /* マーカーを表示 */
 97:                 inkey = getch();
 98:                 mark(*n, 0);                    /* マーカーを消去 */
 99:                 switch(inkey) {
100:                 case 'h':
101:                 case '4':                               /* 左に移動 */
102:                         *n = (n2x(*n)==0) ? *n+6 : *n-3;
103:                         break;
104:                 case 'l':
105:                 case '6':                               /* 右に移動 */
106:                         *n = (n2x(*n)==2) ? *n-6 : *n+3;
107:                         break;
108:                 case 'k':
109:                 case '8':                               /* 上に移動 */
110:                         *n = (n2y(*n)==0) ? *n+2 : *n-1;
111:                         break;
112:                 case 'j':
113:                 case '2':                               /* 下に移動 */
114:                         *n = (n2y(*n)==2) ? *n-2 : *n+1;
115:                         break;
116:                 }
117:         } while(inkey!=0x0d && inkey!=0x20);    /* CRかSPACEが押されるまで */
118: }
119:
120: void moveman(int pl)                    /* 人間側の着手 */
121: {
122:         int n, m, count;
123:         int tmpboard[9];
124:
125:         for(n=0, count=0; n<9; n++)
126:                 if(board[n]==pl)
127:                         count++;
128:         n = 4;                                  /* 初期位置は盤の中心 */
129:         if(count<3) {                                   /* 3個目まで */
130:                 do {
131:                         select(&n);                     /* どこへ置くか? */
132:                 } while(board[n]!=0);
133:                 board[n]=pl;
134:         } else {
135:                 while(1) {
136:                         do {
137:                                 select(&n);             /* どこから? */
138:                         } while(board[n]!=pl);
139:                         put(n, MARK);
140:                         m = n;
141:                         select(&n);                     /* どこへ? */
142:                         if(board[n]==0) {       /* 移動範囲と千日手の判定 */
143:                                 board[m]=0;
144:                                 board[n]=pl;
145:                                 if((prev[0]!=prev[4]||prev[1]!=bd2num(board))&&
146:                                  abs(n2x(m)-n2x(n))<2 && abs(n2y(m)-n2y(n))<2)
147:                                         break;
148:                                 board[m]=pl;
149:                                 board[n]=0;
150:                         }
151:                         puts("¥a");                     /* ベルを鳴らす */
152:                         put(m, pl);
153:                 }
154:         }
155: }
156:
157: int iswin(int *bd, int pl)              /* 勝利条件を満たしているか? */
158: {
159:         int n, c, tn[3];
160:
161:         for(n=0, c=0; n<9; n++)
162:                 if(bd[n]==pl)
163:                         tn[c++] = n;
164:         if(c<3)                                 /* 駒の数が3より少ない */
165:                 return 0;
166:
167:         if(tn[1]-tn[0]==3 && tn[2]-tn[1]==3)    /* 横に並んでいるか? */
168:                 return 1;
169:         if((tn[0]%3)==0 && tn[2]-tn[0]==2)      /* 縦に並んでいるか? */
170:                 return 1;
171:         if(tn[0]==0 && tn[1]==4 && tn[2]==8)    /* 斜めに並んでいるか? */
172:                 return 1;
173:         if(tn[0]==2 && tn[1]==4 && tn[2]==6)    /* 同上 */
174:                 return 1;
175:
176:         return 0;
177: }
178:
179: void copyboard(int *b1, int *b2)        /* 盤データをcopyする */
180: {
181:         int n;
182:
183:         for(n=0; n<9; n++)
184:                 b2[n] = b1[n];
185: }
186:
187: int movecom(int *bb, int pl, int depth) /* コンピュータ側着手決定ルーチン */
188: {
189:         int n, m, count;
190:         int tmpboard[9], bestboard[9];
191:         int best, value;
192:
193:         if(depth>=4)                    /* 4手読み */
194:                 return 0;
195:
196:         best = -pl;
```

縦・横・斜めに一列に並べたほうが勝ちです．

交互に着手し，まず先手は9個のマス目のどこかに駒を置きます．続いて後手が1つ駒を置きます．3個の駒を置きつくしたら，次はお互いに盤上の自分の駒をどれか1つ，縦・横・斜めの空いたマス目に1目だけ動かすことができます．

千日手になったときは，先に動かしたほうが別の手を指さなければいけません．

プログラムを起動すると，先手の人間側の入力待ちとなります．赤い四角のマーカーが盤の中央に表示されているので，駒を置きたい場所に2468かhjklのキーを使ってマーカーを移動し，リターンかスペースを押してください．盤上の駒を動かすには，同様の方法で動かしたい駒を選択してから(選択した駒は青くなります)，となりの空いているマスへマーカーを移動し，リターンかスペースを押します．このとき駒が移動できない場所を選択したり，千日手となる手を打とうとするとベルが鳴って駒の選択に戻ります．

後手は，コンピュータが受けもちます．ゲームは，人間かコンピュータのどちらかが勝つまで続きます．

プログラムの説明

6-44行

Turbo CとMS-CまたはQuickC

```
197:         for(n=0, count=0; n<9; n++)      /* 自分の駒が盤上にいくつあるか */
198:                 if(bb[n]==pl)
199:                         count++;
200:
201:         for(n=0, m=0; n<9; n++) {
202:                 if(count<3) {            /* 盤上の駒がまだ3に満たない */
203:                         if(bb[n]!=0)
204:                                 continue;
205:                         copyboard(bb, tmpboard);
206:                         tmpboard[n]=pl;
207:                 } else {                          /* 盤上の駒は3つ */
208:                         if(bb[n]!=pl)
209:                                 continue;
210:                         if(bb[m]==0 && abs(n2x(m)-n2x(n))<2 && abs(n2y(m)-n2y(n))<2) {
211:                                 copyboard(bb, tmpboard);
212:                                 tmpboard[n]=0;
213:                                 tmpboard[m]=pl;
214:                                 if(++m>=9)
215:                                         m=0;
216:                                 else
217:                                         n--;
218:                         } else {
219:                                 if(++m>=9)
220:                                         m=0;
221:                                 else
222:                                         n--;
223:                                 continue;
224:                         }
225:                 }
226:                 if(depth==0 && prev[0]==prev[4] && prev[1]==bd2num(tmpboard))
227:                         continue;                /* 千日手は避ける */
228:                 if(iswin(tmpboard, pl)) {        /* 直接勝てる手か? */
229:                         best = pl;
230:                         copyboard(tmpboard, bestboard);
231:                         break;
232:                 }
233:                 value = movecom(tmpboard, -pl, depth+1);
234:                 if(value==pl && best!=pl || value!=pl && best==-pl) {
235:                         best=value;
236:                         copyboard(tmpboard, bestboard);
237:                 }
238:         }
239:         if(depth==0) {
240:                 copyboard(bestboard, bb);
241:         }
242:         return best;
243: }
244:
245: void main()
246: {
247:         int i, n, pl;
248:
249:         graphinit();                              /* 画面モードの初期化 */
250:         showboard();                                      /* 盤の表示 */
251:
252:         for(pl=1; !iswin(board, -pl); pl=-pl) {
253:                 if(pl==1)
254:                         moveman(pl);
255:                 else
256:                         movecom(board, pl, 0);
257:                 for(n=0; n<9; n++)                        /* 駒の表示 */
258:                         put(n, board[n]);
259:                 for(i=0;i<4;i++)                          /* 千日手判定用 */
260:                         prev[i] = prev[i+1];
261:                 prev[4] = bd2num(board);
262:         }
263:         printf("¥x1b[15;1H%s won!", pl==1 ? "I" : "You" );
264:         getch();
265:         graphdeinit();                            /* 画面モード復帰 */
266: }
267:
```

ではグラフィック関数に違いがあるため，それぞれのライブラリに依存するルーチンを分けて書いています．関数名も#define文によってTurbo Cのライブラリ関数名に合わせています．グラフィック初期化ルーチンは，これまでにも何度か使用したものをそのまま使っています．

46-51行

各種定数の定義．画面に表示されるマス目や駒の大きさを定義しています．

52-53行

プログラム中，マス目は左上から右下に向かって0～8の番号をつけて表しているので，横位置および縦位置への変換式をマクロで定義しておきます．

54-56行

盤のデータおよび千日手判定用のデータ格納領域は0で初期化しています(static宣言を利用してもよい)．もしmainルーチンで繰り返しゲームを行うようにするなら，board[] およびprev[] は初期化しなおす必要があります．ctable[] には，put () で使う駒の色データが定義してあります．

showboard()

盤を表示します．マス目の数だけ四角形を並べているだけです．

bd2num()

このプログラムでは，千日手を判定するために現在の盤から4手前の状態までを保存しています．マス目は9つで状態は3種類 (空，赤，白) なので，盤の状態はせいぜい3の9乗＝19683通り (実際はもっと少ない) で表せることになります．これならint型に収まり，比較も楽になるので，bd2num () によって盤の状態からint型の数値への変換を行います．

put()

駒を表示します．第2引数の値によって白の駒，赤の駒，マークされた駒，駒のクリアの4通りの表示を行います．

mark()

第2引数が1ならマーカー (赤い四角)を表示し，0なら消去します．

select()

キー入力によって場所を選択するルーチンです．

moveman()

人間側の着手ルーチンです．

125-127行

盤上に自分の駒がいくつあるかを数えます．

128行

select () に渡す初期位置を4 (盤の

中心）に設定します．

129-133行

盤上の駒の数が3より小さいときは，空いた場所どこにでも自由に駒をおくことができます．

134-154行

盤上の駒がすでに3個になっているときは，どれかの駒を選択し動かさなければいけません．駒の選択と移動先を入力し，その移動が正しいか（となりの空いたマス目への移動か），千日手でないかを判定し，まちがった移動や千日手ならベルを鳴らして駒の選択に戻ります．

iswin()

引数として与えられた盤の状態と手番側が勝利条件を満たしているかを返す関数です．まず駒の位置を調べ，3つ未満なら偽（0）を返します．次に縦，横，斜めに並んでいるかどうかを調べ，並んでいれば真（1）を返し，どれにもあてはまらなければ偽を返します．

copyboard()

盤データのコピー用関数です．

movecom()

コンピュータ側の着手ルーチンです．再帰的に呼ばれて先読みをします．引数のdepthが読みの深さを示します．戻り値は盤の状態の評価で，引数の手番(pl)と同じ値を返せば手番側の勝ち，0では形勢不明，相手側の手番(-pl)の値を返せば手番側の負けと，3種類の値をとります．

193-194行

読みの深さを決めます．ここでは，5手以上先では無条件に0を返しているので4手読みということになります．

196行

変数bestには，最善の手の評価値が入ります．初期値として最悪の状態(相手の勝ち）である（-pl）を入れておきます．

197-199行

盤上にある自分の駒を数えます．

201-240行

盤のそれぞれの位置（n）について，そこに駒が置けるか（盤上の駒の数が3未満のとき）またはそこの駒が動かせるか（駒の数が3のとき）を調べていきます．

202-206行

盤上の駒が3個未満のときは，その場所（n）が空いていれば駒を置いてみます．

207-225行

盤上の駒が3個のときは，その場所（n）に自分の駒があるかどうかを調べ，もう1つの変数（m）を動かすことによってさらにどこに動かせるかを調べます．動かせるならば，実際に駒を動かしてみます．

226-227行

千日手となる手は除外します．

228-232行

その手を打って勝利条件を満たすなら，それ以外の手を調べる必要はないのでループから脱出します．

233行

再帰的にmovecom()を呼んで盤の状態を評価します．

234-237行

評価値が自分の勝ち(pl)でbestがplでない場合や評価値が0か（-pl）でbestが（-pl）のときは，bestを更新して盤の状態を保存します．

239-241行

先読みとして呼ばれているときを除き，盤を最良の手を打った状態にして返します．

242行

戻り値として評価の最高値を返します．

main()

249-250行

画面モードを初期化し，盤面を表示します．

252-262行

ゲームのメインループです．plの値を1と（−1）に交互に変えながらループし，ここではplが1なら人間の着手ルーチンを呼び，（−1)ならコンピュータの着手ルーチンを呼んでいます．先手後手を逆にしたければ，ここで着手ルーチンを逆に呼ぶようにすればよいのです(具体的には，253行目と263行目に含まれる（pl==1）を（pl==-1）に変更する)．

263-265行

勝者を表示し，キー入力待ちをしてから画面モードを復帰します．

ワンポイントテクニック

コンピュータの思考ルーチンについて‥‥先読みと盤の評価

今回とりあげた「三子棋」というゲームは○×（マルバツ）ゲームと似ていますが，3個ずつ打ち終わったあとは盤上の駒を動かすというルールでかろうじてゲーム性を保っています．それでも千日手の判定を厳しくしないかぎり，まず勝負はつきません．

このような盤と駒を使うゲームでは，画面表示や入力にもある程度の行数がかかってしまうので，限られた行数でプログラムするとなると単純なゲームしか題材として選べず残念ですが，基本的な構造はオセロやチェスのような，より複雑なゲームの場合も同じです（"強い"思考ルーチンを作るのはたいへんですが‥‥）．ここでは，その思考ルーチンの基本的な構造について解説することにします．

思考ルーチンの目的は，「よい（できれば最善の）手を探すこと」です．そして，それを実現するために「先読みと盤の評価」を行います．先読みとは，仮想的な盤の上で手を進めてみることで，盤の評価は形勢判断です．

先読みは機械的な仕事で，ルールにしたがって動かせる手をすべて指してみることの繰り返しです．手を見落とすこともなく，正確に先読みできるコンピュータの得意とするところです．

盤の評価は思考ルーチンの核ともいうべき大事な部分で，ある盤の状態がどちらにとってどのくらい有利な形勢かを判断し，点数をつけて数値化します．この形勢判断をいかにうまくするかが思考ルーチンの強さを決めるのです．

今回のプログラムでは，盤の評価を単純に「勝ったか，負けたか，勝負がまだついていないか」という3つの値で決めています．もちろん，これは三子棋が数手で勝負がつく（こともある）ゲームだから有効な方法であって，通常のゲームでは，なんらかの方法で「勝負がついていない」状態の形勢を数値化することになります．

とりあえず先読みと盤の評価の方法が決まったので，あとはこれらを組み合わせるだけです．現在着手可能な（打つことのできる）手のうち，もっとも点数の高い手を打てばよいというわけです．しかし，1手読みならそのまま1手打った状態の盤の評価値を比べればすみますが，2手以上先の盤面を評価した場合，どのように値をくらべればよいかが問題となります．基本的には，お互いに自分の有利な形勢になるように手を打つと考えて，手を絞っていくことになります（これ以外の戦略をとることもある）．つまり，N手読みならば，ある1つの（N－1）手目まで打った局面からN手目の手番のプレイヤーがもっとも（彼にとって）評価値の高くなるような手を打つと考え，その評価値をその（N－1）手目の評価値として繰り上げます．これを（N－1）手目から1手目まで繰り返して，1手目の中で評価値がもっとも高くなる手を採用するという手順です．

できるだけ先読みは深いほうがよいのですが，無限に先読みをするわけにもいかないし，たとえ有限の手数で終わるゲームだとしても，すべての手を読むには天文学的な時間がかかるのが普通です．最初から勝負がつくところまで完全に読み切ってしまえれば真の最善手が得られますが，そのゲームは先手勝ちか後手勝ちかが決定してしまうわけですから，すでにゲームとしてのおもしろさは失われてしまいます．幸いにして，オセロのように一見単純に見える（たかだか60手で勝負がつく）ゲームも現在のコンピュータの力では最後の10数手～20数手を読み切るのが精いっぱいといったところなので，心配することはないようです．

ここで説明した手の決定法（min-max法）は，そのままでは無駄が多い手法で，より強い思考ルーチンを作るには，先読みの途中でそれ以上先を読んでも意味のない手を捨てたり（α-β枝刈り），途中で形勢を評価して手を絞り込んだりといった工夫が必要になります．プログラムの書き方も速度優先にして時間を節約し，少しでも深く先読みをするようにします．

今回のプログラムでは先読みを4手読みにしてありますが，これだと実は簡単に勝つ方法があるのです．まず白（あなた）は中央に打ってください．すると赤は左上に打つので，白は右下に打ちます．赤はここで左中央に打つので，白は左下に打ちます．これで，赤はどこに打とうとも白の勝ちです．負けを覚悟するともっとも右下に打つようになっているので，このあと赤右中央，白中央から中央下と続きますが，もしここで赤が中央下に割って入っても白右下から右中央で白の勝ちは変わりません．

ここで赤が敗着手を打ってしまったのは，4手先の時点までしか先読みをしていないので，白の勝ちを予測できなかったからです．ですから，リスト1の193行目の（depth>＝4）を（depth>＝6）にして6手先まで読むようにすれば，赤は左下に打って白の勝ちを防ぐはずです．そのまま6手読みにすると，少し遅く感じるかもしれませんが，無駄な処理（bestboard[]はdepth＝＝0のときだけ転送すればよい，など）を削れば速くなります．

リスト1の思考ルーチンは，そのほかにも同じ評価値の手が複数あると必ず左上の手を優先して打つので変化に乏しかったり，4手を越える繰り返しパターンによる千日手を判定していないなどの欠点があります．前者は乱数によって手を選ぶようにしたほうがおもしろいし，後者はもっと千日手判定用データを長くとって比較するようにするなどとするべきでしょう．

パソコン博物館 VII

世界を制覇したパソコン

IBM PC

IBM社 IBM PC

井戸禎光

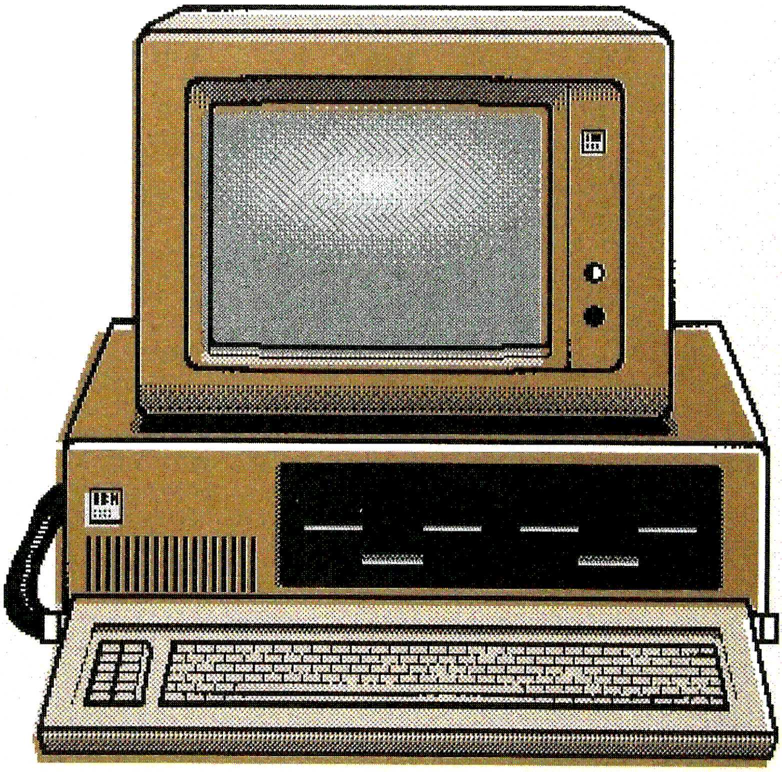


Illustration
TOSHIHITO ITO

1974年のアルテア8800以来，パソコン界には多くのメーカーが参入し，技術，マーケティングなど多くの面で競っていた．しかし，IBM 社が参入すると数社の例外を残してほとんど消えてしまい，日本をのぞいた全世界が統一されてしまった．この理由は何であったのか．

残った数社とは，アップル，タンディ，コモドールなどであり，日本の場合は漢字という障壁に守られて，NEC 他のメーカーが残っている．

IBM PC は，だれでも入手できる部品のなかで最高のものを組み合わせたもの(CPU はインテル，OS はマイクロソフト，ディスクはタンドン，プリンタはエプソン)に過ぎないという人もいる．しかし，組み合わせの選択肢はひとつではなかったし，他社でできなかったことも事実である．IBM 社は大型汎用機の世界では BIG BLUE と呼ばれ，コンピュータと同義語の時代もあった．その会社で作られたのが IBM PC である．

IBM PC の特徴は，その開発責任者であったドン・エストリッジ氏によって決められた以下の3項目になろう．第一に，部品は社外から調達する（オープンアーキテクチャが可能になる）．第二に，当時主流の8ビットプロセッサは使わず16ビットプロセ

ッサを使う．第三に，当時主流であったBASICは使わずOSを使う．

プロジェクトは1980年8月に発足した．メンバーはわずか12人であったが，発売時の1981年8月で200人程度，1984年には9500人にもふくらんだ．

成功の最大の理由は，「オープンアーキテクチャ」であるといわれている．要するに「利益を独り占めにしようと試みると失敗する」ということであろう．ソフトメーカー，ボードメーカー，システムメーカーなどが，協力しあって業界作りをしないと大成功は難しい．

第一の特徴である部品の社外調達については，社内的には難しい面がある．大手企業の中で自社製品を使わずにすますとなるとたいへんであろう．実際，彼が決断した項目は，研究開発，ソフト開発，ハード開発，マーケティング，広告，販売，製造，財務など多岐にわたっている．通常は，このような決定はほかの部門の人が行うことが多く，これでは仕事が止まるだけで進むことは希である．ところがこの場合は，会議に次ぐ会議で仕事が止まるということがないように，経営トップからの直接の指示であったので，中間管理職や専門機能組織が介入する余地がなかったといわれている．これは企業内ベンチャーの成功例である．しかし，ベンチャーに入らなかった人が大勢いるわけであるから，そのねたみが鬱積してくる．これをうまく取り除いておかないと，足を引っ張る者がおおぜい出て，結局のところ失敗に通ずることになりかねない．

IBM PC 追加ボード用ピン信号リスト

GND	b1	a1	I/O CH CK*
RESET	b2	a2	D7
+5V	b3	a3	D6
IRQ2	b4	a4	D5
-5V	b5	a5	D4
DRQ2	b6	a6	D3
-12V	b7	a7	D2
(NOT USED)	b8	a8	D1
+12V	b9	a9	D0
GND	b10	a10	I/O CH RDY
MEMW	b11	a11	AEN
MEMR	b12	a12	A19
IOW	b13	a13	A18
IOR	b14	a14	A17
DACK3*	b15	a15	A16
DRQ3	b16	a16	A15
DACK1*	b17	a17	A14
DRQ1	b18	a18	A13
DACK0*	b19	a19	A12
CLK	b20	a20	A11
IRQ7	b21	a21	A10
IRQ6	b22	a22	A9
IRQ5	b23	a23	A8
IRQ4	b24	a24	A7
IRQ3	b25	a25	A6
DACK2*	b26	a26	A5
T/C	b27	a27	A4
ALE	b28	a28	A3
+5V	b29	a29	A2
OSC	b30	a30	A1
GND	b31	a31	A0

一方，このプロジェクト組織の中で働く人のモチベーションはどうだったか．ベンチャービジネスでは，事業をはじめた当初に買った株が数百倍にもなって，金銭的な報酬を得る可能性にかけて働くのであろうが，IBMのような大企業では無理である．この場合は「新しいやりがいのある仕事に参加して，自分の考えで社内の仕事を動かしていける」ということであったといえよう．通常，大手企業では各人が考えて作業していたのではまとまりがつかなくなるので，各人に与える裁量の範囲を狭めざるをえない．

第二のCPUについては，当時では一番進んでいたインテル社の16ビットCPUである8088を使った．8ビット機の限界がすでに読めていたのである．

第三のOSは，マイクロソフト社に頼んで新しく作ってもらった．これが後々まで16ビットPC用OSの中心となるPC-DOS(MS-DOS)である．当時もOSはあったが，それは8ビットでCP/Mと呼ばれ，ゲイリー・キルドール氏のデジタルリサーチ社の製品であった．しかし，何かの行き違いから同社には発注されずマイクロソフト社が作ることになった．マイクロソフト社のほうは，シアトルコンピュータプロダクツという会社のもっていたOSを買い，CP/Mとアッパーコンパチにして IBM社に収めることができた．これが1981年8月である．

以上が成功例であるが，IBM社といえどもすべての機械で成功してきたわけではない．たとえば，1983年に発表したPC Jr.，1987年のPS/2などである．前者は，DMA，キーボード，メモリ容量などに欠陥があるし，後者の場合はOSであるOS/2および追加ボードのコネクタ部分の規格であるマイクロチャネルの公開を渋ったりして評判を落としている．

IBM PCの生みの親であるドン・エストリッジ氏は，テキサス州ダラスフォートワース空港におけるデルタ191便トライスターの事故で47歳の若さで亡くなってしまったが，彼の作品であるPC/ATは，いまなお世界の標準機としての地位を保っている．

祐 安 重 夫

プログラマの道具学

第7回 酒

仕事が一段落した夕方，はやめに仕事場をでて，開店したばかりのバーのカウンターで，今日の最初の1杯を何にするかかんがえるのは，なかなか気分のいいものである．夏のあつい日なら，最初の1杯はギムレットのようなカクテルにすることもあるし，シェリーもわるくはない．あるいは最初から，つよい酒を生のままでやりはじめることも，結構ある．ただし自分からすすんでウィスキーをのむことは，あまりない．

なぜか日本では，バーやスナックのようなところで，カラメルで着色し化学調味料で味つけした国産ウィスキーの水割をのむことが，妙にはやっているようだ．それもやたらと水道の水でうすめ，製氷機か冷蔵庫でつくった氷をいれた代物である．本物の酒のみ以外の人間と，つきあいで酒をのむということになると，そういうところにつれていかれる可能性がたかいので，できるだけさけるようにしている．うっかりつきあう羽目になると，翌日はたいていひどい二日酔になやまされ，仕事にならないことになる．

酒をのむことは，文化の領域に属する行為である．もちろん，こういったからといって，酒をのまない人をおとしめるつもりはない．酒をのまないという選択も，また文化の問題だからである．おとしめるべきは，水割と称する代物をガブのみしている連中ということになるだろうか．

もともとウィスキーというものが，あまり面白味のある酒ではないのかもしれない．もっとも本家のスコッチにはさすがに化学調味料ははいっていないので，モルトを生のままでやるのはわるくないし，バーボンをオン・ザ・ロックにしたり，それに少量の炭酸をそそいでのむのも一興ではある．もちろん上質の水とまっとうな氷をつかうなら，水割もうまいものではある．

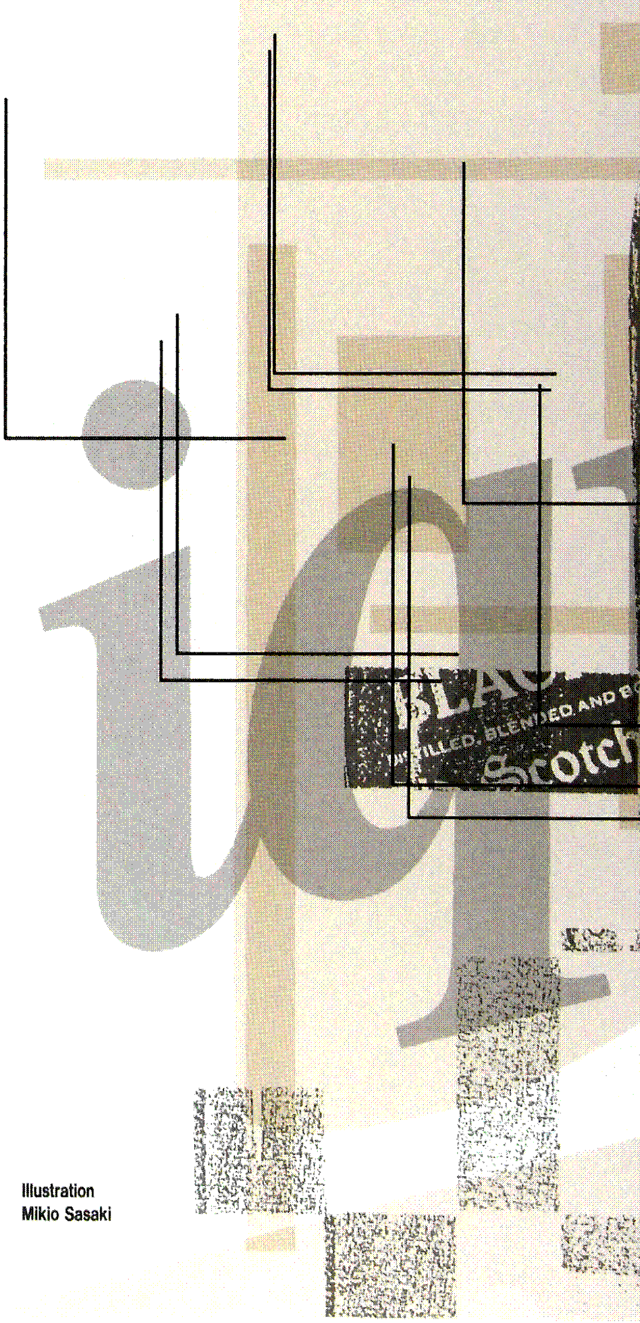


Illustration
Mikio Sasaki



ビールやワインは，最初の1杯にはならない．ワインは食事ときりはなしてかんがえることができないし，ビールはといえば昼食にそえるか午後のお茶がわりというのが，ふさわしいような気がする．日本酒も料理ときりはなすのがむずかしそうだし，昼さがりにそば屋でそばを肴にやるのが，私には一番ふさわしいようだ．もっとも昼間からのむことは，それほどはないが．

しかし，本当にたのしんでのむ酒は，日本でいえば愚劣な税制によって乙類などとよばれている本格焼酎であり，中国で「白酒」báijiǔ，英語でspiritsとよばれるような，蒸留酒の類である．ありふれたところではウォッカやジン，ラムといった，カクテルの材料によくつかわれるようなものがあるが，これらはストレートやオン・ザ・ロックにした方が，よりたのしめる．

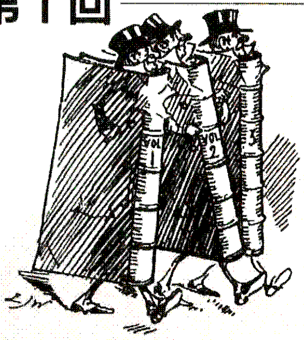
さらにいいのは，アクアビットやキルシュといった酒である．香草や果物の風味がほのかにのこっており，すこしづつ味わいながらのむには，最適である．このような酒をのんでいるかぎりは，あまり翌日まで二日酔がのこることがないのも，なにやら不思議ではあるが，結構なことである．

あるいは林檎からつくったブランディである，カルヴァドスもわるくない．普通の葡萄からつくったブランディにくらべると，さわやかでフルーティな香りがあり，なかなか気にいっている．ちなみに広域Unixネットワーク上での私のところのドメイン名は，calvados.juice.or.jpである．

原稿もできたし，そろそろ今日の最初の1杯をかんがえる時間である．

IC千夜一夜

第1回 IC技術の基礎知識



小型化，高速化，そして省電力

虎岩 登

このコーナーでは，パーソナルコンピュータを支える部品たちの話をしていきたいと思います．CPUの歴史を中心にしますが，メモリや周辺デバイスをとりまぜて，技術の推移がわかるようなものになればと考えています．

もちろん，その多くが活躍したのははるか昔のことです（インテル社が4004を発表してから今年で20周年になりますね）．いまさら知ってどうするという声もあるかもしれませんが，いまだに変わっていないところもありますし，歴史背景を抑えておけば現在の技術を理解しやすくなります．

今回は，基礎知識に相当するところをおさらいしましょう．マイクロコンピュータのような特定のLSIについてではなく，ICの技術の発展について見ていきます．

規格の標準化でLSI開発に大きな影響

TTL

周辺機器とのインターフェース部分のプログラムが得意な方ならご存知でしょうが，コンピュータのハードウェア仕様書とか，何とかインターフェースといった規格にTTLレベルがどうのと，いまでも書かれています．それ以前のリレーや真空管など，個別の部品を使ったトランジスタやダイオード回路でもコンピュータを作ることはできますが，ここでは触れません．

TTLというのは，Transistor-Transistor-Logicの略ということになっています．なぜ，トランジスタが2段重ねになっているか．じつは，これ以前の1960年代半ばにDTL (Diode-Transistor-Logic) というものが存在したからです．DTLの基本的な回路を図1に示します．入力に近いところにダイオードが使われて，後ろにトランジスタ回路がつながっているのでDiode-Transistorなわけです．入力のダイオード回路でANDやORなどの論理演算を行います．ダイオード回路だけではNOT演算はできないのと増幅作用もないので，必要があるときは，トランジスタ回路でNOT演算を行いながら出力電流を大きく取れるようにしています．TTLは，図2のように最初のダイオード回路をトランジスタ回路に置き換えて，ANDやORもトランジスタだけで行います．だからTTL．

いまはTTLという言葉は聞いても，DTLというのは聞きませんね．デジタ

ル回路がIC化されはじめた初期のころはDTLが使われていました．TTLがこれほど普及したのには，動作速度などの特性が改善されていたこともありますが，それよりも，開発元のテキサス・インスツルメンツ社がシリーズ化を進め，統一したインターフェースで，さまざまな機能ブロックをIC化していき大ファミリーを形成したことがあげられます．しかも，多くのメーカーがライセンスを受けて同じ規格のICを製造したのです．ちょうど今日アメリカのほうで，IBM PC系列のパーソナルコンピュータがコンパチ機を従えて主流になっているのとよく似ていますね．

ICといえど，統一した規格で多くの会社から同じものが供給されれば，ユーザーは安心して使えるわけです．このときに決められた5Vという電源電圧や，入出力のHレベルやLレベルの電圧などの規格は，その後のLSI開発にも強く影響を与え，できる限りTTLと同じ電圧規格になるように設計されてきました．

ユニポーラトランジスタの利用

p-MOS LSIとn-MOS LSI

TTLのほかに，ふつうにいうトランジスタでできたロジックICには，ECL (Emitter Coupled Logic) というのもあります．超高速論理回路を構成するために使われるのですが，コストや集積度，使い方の難しさから，大型コンピュータには使われてもパーソナルコンピュータでは一部にしか使われていないので詳しい紹介は省きます．実は後述する理由から，パーソナルコンピュータレベルでは使う理由がますます低くなっていることもあります．

先ほど「ふつうにいうトランジスタ」といういい方をしました．それは，n型半導体とp型半導体が，NPNとかPNPとかいうかたちにサンドイッチになっている構造のものを指します．n型とp型の2つの極性をもっているので，バイポーラ (Bipolar) 型とう呼び方をします (図3参照)．このあたりの話は，NHK特集の「電子立国日本の自叙伝」という番組でご覧になった方も多いでしょう．

さて，バイポーラに対してユニポーラ (Unipolar) トランジスタというものもあります．Uniとは1つのという意味ですから，p型かn型どちらか一方しか利用しないトランジスタという意味ですね．動作原理からいうと，電界効果型トランジスタ (FET) というものがそれにあたります．FETにもいろいろあるのですが，デジタルICで用いられることが多いのは，MOSという略号を頭に付けたMOS-FETです．MOSとはMetal-Oxide-Semiconductorの頭文字を集めたもので，金属―酸化物―半導体と，本当にこの順に重なっているという構造を指します．余談ですが，金属と半導体とあったら，ふつうはこれに絶縁体(insulator)という言葉が並ぶはずです．ちょっと専門的な説明になりますが，MOSというのはMISの特別な場合なのです．MISというのは，もちろんMetal-Insulator-Semiconductorの略です．べつに絶縁体は酸化物である必要はなくて，窒化物などが提案されて，特有の性質をもっています．なにぶんにもシリコン製の半導体では酸化膜は作りやすい上に安定していますから，絶縁体の代表格として酸化物が広く使われているわけです．

このMOS-FETないしユニポーラトランジスタは，構造が簡単なので集積度を上げやすい．つまりLSI向きの性質をもっています．現在のLSIというのは，ほとんどMOS-FETが使われています．もっともMOS-FETにも種類があります．チャンネルというFET動作の中心部に，p型半導体を使うかn型半導体を使うかということなので，p-channel MOS-FETとかn-channel MOS-FETとかいわれます．簡単にするため，p-MOSFETやn-MOSFETと表記します．p-MOS-

図1　DTLの回路図

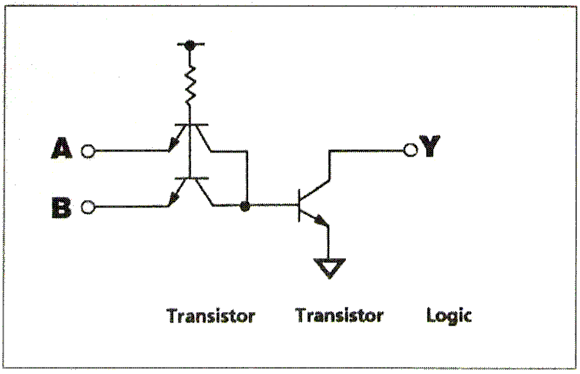


図2　TTLの回路図

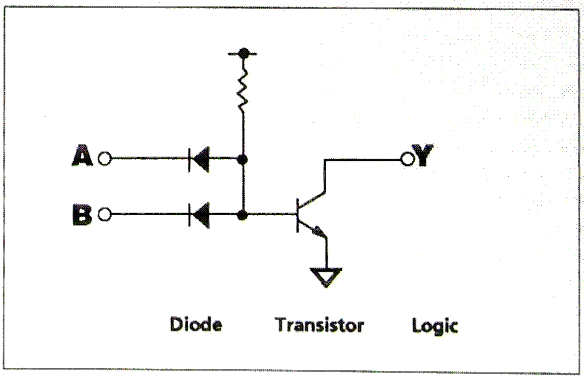
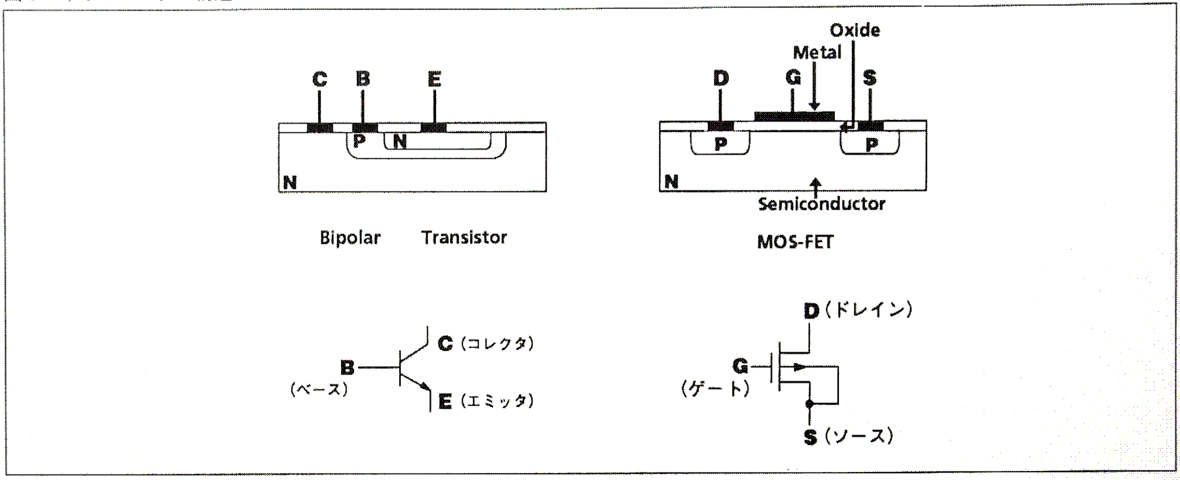


図3　トランジスタの構造



FETを全面的に採用したICは，p-MOS ICとかp-MOS LSIとか呼ばれ，製造しやすいことから初期の電卓用LSIやマイクロコンピュータにも用いられていました．たとえば，インテル社の4004や8008などがそうです．ただ，電源電圧が15V前後必要で，すでに大量に利用されていたTTL ICと接続しにくかったり，スピードが遅いなどの理由によって，1970年代半ばからコンピュータ用の主流はn-MOS LSIに移っていきました．現在でも，後述するC-MOSのLSIだと，とくにことわっていなければ，まずn-MOS LSIだと考えていいでしょう．n-MOSFETは，p-MOSFETより小型にできるというのが変化のおもな理由だと思います．

構成要素のトランジスタが小型になるというのは，ICにとって重要なことです．ただ単に集積度が上がって，ひとつのICに多くのトランジスタが載せられるというだけでなく，小型になると動作速度が速くなるからです．単純に考えても，半導体の中を電子が動く速度が一定なら，小さくなったほうが高速になるといえるでしょう．実はほかにも理由があるのですが，まずは素子を小型化するというのは動作速度の改善に直接関係することだと覚えておいてください．

低消費電力の要求に応えて C-MOS ICの登場

さて，デジタル回路にも低速でもいいから低消費電力であってほしいという要求がありました．たとえば，デジタルの腕時計用LSIなどです．また，小型化競争をしていた電卓などにも同様の要求がありました．あまりに小型化薄型化していくと，電池を入れるスペースが少なくなってきます．小型の水銀電池などは容量がありませんから大食らいのLSIではすぐに使えなくなってしまいます．うまく太陽電池でまかなえるだけの消費電力にすれば，電池を交換できるような構造にしなくてすむうえに，太陽電池は薄い板状のものですから，極限まで薄くすることが可能になります．

そんなわけで，低消費電力を売りものにしたICが誕生しました．それがC-MOS ICと呼ばれるものです．これは，p-MOSFETとn-MOSFETを必ずペアにして内部の回路を構成するのが特徴です．必ずペアにしますから，n-MOS ICではトランジスタ1個ですますようなところも2個のトランジスタが必要になって，集積度は上げにくいのですが，静的な消費電力がほぼ0にできるという特徴があります．静的というのは，出力が0か1のまま，変化していない状態のことを指しています．逆にいえば出力が変化する瞬間にはわずかですが電力を消費してしまいます．

このC-MOS ICはTTL ICと同じように，いろいろな種類の論理回路をシリーズ化してRCAとモトローラが売りだしたため，70年代半ばから広まりはじめました．最初のころには，さまざまなメーカーが独自の規格で売りだしたのですが，この2社の連合軍に事実上飲み込まれてしまいました．TTL ICのシリーズのことを，74シリーズなどと呼ぶことがあります．型番が74××というふうに74ではじまるからです．RCAのC-MOS ICは，40××だったので40シリーズのC-MOSということもあります．

70年代から80年代の頭までは，TTL ICの改良版であるLS-TTL ICやS-TTL ICと，この40シリーズのC-MOS ICが汎用の小規模ICとして栄華を誇っていました．マイクロコンピュータのCPUやら周辺LSIのまわりには，必ずこれらのICがたくさん使われていました．並みのマイクロコンピュータ用回路にはLS-TTL，とくに

高速のものが必要なところにはS-TTL，低速でもいいからバッテリーバックアップなどが必要で，低消費電力が要求されるところにはC-MOSという具合に使い分けされていました．

そのころからC-MOSが進化しはじめます．C-MOSでもトランジスタをできるだけ小型にすれば，静的な低消費電力という特徴は残したまま高速動作が可能になります．動作周波数さえあまり高くなければ，まだTTLなどほかの回路方式より低消費電力であったりします．低消費電力ということは発熱が少ないということでもあります．LSIも集積度が高くなり，消費電力が大きいままだと放熱に困るので，C-MOS化が進む傾向にあります．今開発されているものは，ほとんどC-MOS LSIが主流といってもいいかもしれません．

ICには，どうしてもトランジスタを小さくできないところがあります．とくに，ICの出力用トランジスタは，外部回路に対して十分な電流を供給しなければならないので，小さくはできないのです．すなわちICの外部に信号を出すと遅くなる，ペナルティがあるということです．逆に，すべて１つのICで完結させれば，高速の内部回路用トランジスタを最大限に利用できるということになります．

そうなると，TTLやC-MOSの個別の小規模なICで回路を組むのは，速度的に圧倒的に不利になります．コンピュータなどデジタル回路のほとんどは，常に高速化への強い要求がありますから，小規模なIC，とくに単純なゲートIC（ANDやOR演算のような単純な論理演算だけしかできないICだと思ってください）は使われない傾向があります．パーソナルコンピュータのメモリとCPU以外の回路は，ほとんどASIC化が進んでいます（ひと昔前はカスタムLSIと呼んでいました）．周辺回路をすべて取り込んで大規模なLSIにしてしまえば，C-MOSでも並みのECLのゲートICを使うのと同程度の速度が得られるし，またそのような大規模なLSIには，発熱などの点からC-MOSが使いやすいというわけです．

そういえば最新のi486マイクロプロセッサもC-MOSです．ただし，高速動作をさせるのでとくに低消費電力とはいえません．専用の放熱器をつけて空冷してやらなければつらいようです．

TTL系列も進化を重ね，高速低消費電力をめざしてさまざまな改良版が開発されていますが，ASIC化の波はその先を行っています．最近のパーソナルコンピュータの内部を見ると，バッファ回路以外は見あたらなくなっているように見受けられます．バッファ回路とは単純にいうと，信号を増幅して多くの回路に信号を伝えるための回路ですから，LSI化してもメリットがないからなのです．

ICの常識ウソorホント

◆MOS-ICは静電気に弱い？

MOS-ICは静電気に弱いというのは本当でしょうか．MOS-FETの酸化膜層の厚さを約0.1μmと見積もって5Vの電圧を両端にかけたとしただけでも，1cmあたり500kVもの電界が加わっているのと同じですから，さらにけたはずれに高い電圧を加えたら，実際簡単に壊れてしまいます．ただし，それは裸のMOS-FETの話．

現在のMOS-ICには，すべて厳重な保護回路が入出力端子に付けられています．よほどのことがなければ，ICが劣化することはないと思います．ということで，ふつうはあまり神経質になる必要はありません．C-MOS ICのではじめのころは，簡単に壊れたようですが．

◆半導体は熱に弱い？

完全に迷信になっている知識もあります．いわく「半導体は熱に弱い」．ゲルマニウムトランジスタ全盛期には意味があったと思いますが，現在の半導体はほとんどシリコン製ですから，ずっと熱に強くなっています．ゲルマニウムはハンダ付けの熱でも気をつけるべきでしょうが，シリコンはその程度の温度では破壊されることはありません．コンデンサのほうがずっと熱に弱いものが多いのです．

ただし，動作時の熱は別です．半導体は温度によって特性がかなり変化します．また動作時は半導体自身が電力を消費して発熱しますから，パッケージの外側ではたいした温度でなくても，チップは高温になっていたりしますので，放熱に十分注意すべきです．もっとも，こういうことはだいたいはハードウェアの設計者が気をつければすむことなのですが．くれぐれもコンピュータなどの電気製品の吸気孔やファンの出口など塞がないようにご注意を．

MS-DOS TOOL LABO

MS-DOSツール・ラボ

第4回 ファイル検索プログラム T.EXE

加野島英渡

今回は，先月解説した"再帰"を用いた実用的な例として，コマンドラインから指定されたファイルを，サブディレクトリを順々に下降しながら検索するプログラムを作成する．また，動的にメモリ領域を確保する手法のひとつである"スタック"についても，その実現方法を示した．

ディレクトリの中にディレクトリがある

数年前，CP/MからMS-DOSに移ったときに，ファイルリストをとるとその中にファイルリストの集合があるということにショックを受けた．聞いてみればなんのことはない，かのUNIXからいただいたもののようだが，そのUNIXも知らなかったのだから，驚いたって不思議はないのだ（もちろん威張ってするような話ではない）．

先月お約束したとおり，今月はちょっとこのサブディレクトリの中を探ってみることにしたい．

うまい話には裏がある

サブディレクトリというのはたいへん便利なものだ．なにが便利かというと，いくつかのファイルをひとまとめにして，それに名前をつけておくことができる．しかも，これには階層をつけることができる．階層というのは，建物の上の階と下の階のように重なった構造のこと．大分類，中分類，小分類とか，会社だと部があってその下に課があってその下に係があるというような構造のことだが，たくさんのファイルをこうやって細かく分類していくことができるので，うまくやればたいへんすっきりとわかりやすくファイルを整理することができる．

ただ，この「うまくやれば」というのがくせもので，一般的にものごとというのはうまくやれないものなのだ．うまくやれるのが普通だと考えてしまうと，できなくて落ち込むことになる．うまくやれるのは，たまたま偶然が重なっていいほうに動いたか，あるいはもう何度も同じことをやっていて手順

が決まっているときくらいのものだ．そう思っていればまちがいない…と，わかっちゃいるけどやめられないのは植木等さんだけではないようで，そういうことがちゃんとわかっているはずのプロでも，利用者が自分と同じくらいに「うまくやる」ことを期待したプログラムを作ってしまったりする．

ま，それはおいといて，サブディレクトリを作ることも，たいていの場合はうまくやれないことのほうが多い．やたらと深いディレクトリができてしまったり，ルートに100個もサブディレクトリを作ってしまったり．こうなると分類もなにもあったものではない．そういうときに，「この中のどこかにあるはずの，この名前のファイルを見つけてくれ」というお願いのできるプログラムを作ろうというのが今月の趣旨だ．このプログラムで，先月説明した再帰を使う．

どうやってファイルを探すか

先月号で，サブディレクトリのツリー（ちょうど木のように枝分かれしているのでこう呼んでいる）は，再帰的な構造であると書いた．サブディレクトリの中身が，一般のファイルとサブディレクトリでできているというように，自分自身の定義の中に自分自身を含んでいるからだ．そして，再帰的な構造を扱うプログラムは，それ自身が再帰的な構造をもっていることが自然であるとも書いた．

では，自然にやってみることにしよう．このプログラムのメインのロジックを日本語で書くと図1のようになる．順序は逆でもいい（図2）．

自分自身の定義の中で，自分自身を参照していることがおわかりいただけるだろうか．先月号のこの連載記事の中にサブディレクトリのツリー構造の簡単な図があるから，その図と図1，2に書いたプログラムの構造が符合することを確認していただきたい．

再帰にはスタック，しかし……

先月号で，再帰的な構造のプログラムではデータ領域を動的に確保しなくてはならないと書いた．また，それに適した構造はスタックであるとも書いた．あそこでスタックと呼んでいたのは，使い方からわかるとおりCPUスタックのことである．86系のCPUではSP（スタックポインタ）と呼ばれるレジスタが用意されていて，PUSH／POPあるいはCALL／RETなどの命令でデータや戻り番地の出し入れに暗黙に使用されている．このSPを用いたスタックのことをCPUに用意されているスタック，CPUスタックと呼んでいる．とくにことわらない場合は，このCPUスタックを単に「スタック」と呼ぶことが多い．

ところで，スタックというのはデータ構造につけられた名前であって，けっしてSPレジスタや8086の専売特許ではない．ソフトウェアによってスタック構造を作ることはもちろん可能で，CPUスタックに比べてスピードは落ちるものの，データサイズや構造をプログラマが自由に決めることができる．

また，CPUスタックは処理間のデータの受け渡しにはちょっと不便で，呼び出し先に渡したいデータを積んでも，呼び出すときに戻り番地が同じスタックの上にどかんと積まれてしまうので，わざわざそれをよけてデータを取り出さなくてはならない．これも，データ用に独立したスタックをソフトウェアで実現しておけば自由がきく．スタックについては来月取りあげてみたい(よかった，来月のネタができた)．

というわけで，スタックを使うからといって必ずしもCPUスタックのご機嫌をうかがう必要はないので，ここはひとつ自前でスタック構造を用意してやることにしたい．もちろんCPUスタックを使ってもいいのだが，だれでもあたりまえにやることをやっていたのではおもしろくないじゃないか．

さて，スタックを自前で用意することにしたのだが，自前でやるからにはなんでも自分でカタをつけなくてはならない．とりあえずある程度の大きさ

図1

```
サブディレクトリの検索と表示 とは
    サブディレクトリがあるならば
            サブディレクトリの検索と表示 をして
    一般ファイルを表示することである。
```

図2

```
サブディレクトリの検索と表示 とは
    一般ファイルを表示して
    サブディレクトリがあるならば
            サブディレクトリの検索と表示 をすることである。
```

のメモリ領域が必要になるのだが，これをどこからもってきたらいいのか．

メモリ獲得

MS-DOS上で動作するプログラムがメモリを使う方法は2通りある．1つはプログラムのソースであらかじめ領域を定義しておくという方法だ．これには，DBやDWなどの擬似命令を使用する．こうしておけば，アセンブラやリンカがどれくらいのメモリが必要かを計算してくれて，プログラムの起動時にローダ(OSの基本機能のひとつ．指定されたファイルをファイル管理システムの助けを借りて実行可能な形でメモリ上に展開する）によってそのぶんのメモリが自動的に確保される．

もう1つは，プログラムが動作中にダイナミックにメモリを獲得する方法だ．MS-DOSにかぎった話ではないが，システムの資源（主記憶，外部記憶，CPUなど）はOSによって統合的に管理されているので，これにおうかがいをたててメモリを使わせていただく．いらなくなったら速やかにお返ししなければならない．

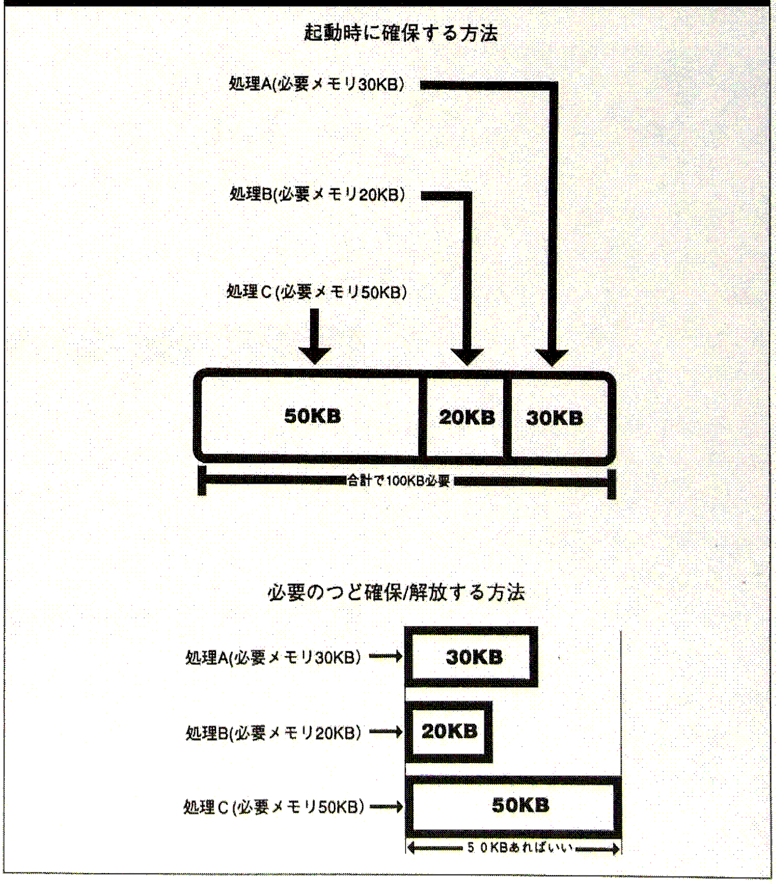
なぜこんなめんどうなことをするかというと，おもにメモリを効率的に使用するためである．

プログラムはその動作にあたってさまざまなメモリ領域を必要とするが，その中には実際にその場面になってみないと，どれくらいのメモリが必要かがわからないというものがある．たとえば，ファイルをいったん全部メモリに読み込むプログラムがあったとする．このとき必要とされるメモリ領域は，そのファイルのサイズによって決まる．こういう領域まで第一の方法のようにあらかじめシステムに自動的に確保してもらうようにするならば，考えられる最大のメモリ領域を宣言しておかなくてはならないが，そんなことは事実上不可能である．できたにしてもメモリを馬鹿食いする困ったプログラムになってしまう．

また，できるだけ多くほしいが，少なければ少ないなりに，多少効率は落ちるものの動作できるというメモリ領域の使い方もある．たとえば，ファイルの読み書きに使うバッファがこれに該当する．ファイルに書き込む場合は，メモリ上のバッファにぎりぎりまでため込んでおいて，いっきにディスクに書き込んだほうが効率がいい．また，読み出す場合でも，とくに連続的に読み出すときなどは，ドサッと多めにメモリに読み込んでおいて，そこから必要なぶんだけ取り出してくるほうが，必要のつどいちいちファイルから読み出してくるよりはスピードが速い．このためのバッファは，一般的には大きいほうがいいのだが，小さくてもそれなりに動作する．

まだある．前の例でもそうだが，プログラムのある場面でのみ必要なメモリ領域というものがあるのだ．たとえば，ファイルを読み込んでなんらかの処理をするプログラムでは，実際に読み込みのためのメモリを必要とするのはその処理の中だけで，それ以前，あるいはそれ以後の処理では必要ない．こういうタイプのメモリを必要とする処理がたくさんあった場合，それぞれが起動時に必要なぶんだけのメモリを獲得するようになっていたらどうなるだろうか．ちょっと図3を見てもらいたい．

図3



メモリの無駄食いは，自分だけが動いているときはまだいいのだが，そこから子プロセスを実行しようとする場合，あるいはマルチタスクのOSで同時にいくつものプロセスが動くような場合には非常に迷惑なものになる.

自分だけが動いているときはまだいいと書いたが，そうとばかりもいえない．たとえば，プログラムが起動されたときにメモリの余裕が60KBであった場合，そのつどメモリを獲得する2の方式なら10KBの余裕をもって動作するが，最初にいっきに獲得する1の方式ではメモリ不足で動作できない．まったく同じことをするプログラムが，そのメモリ獲得のやり方で動作できなくなってしまうこともあるのだ.

もっとも，だからといってすべてのメモリを実行時獲得にするというのはいきすぎである．OSのチェックを受けるためにメモリ獲得にはたとえわずかでも時間がかかるし，獲得できなかった場合の処理などめんどうなことも多い．また，実際に使用する場面になってメモリが獲得できなかったら身動きが取れなくなるようなこともあるので，向き不向きをよく考えて方法を選択しなければならない.

参考までに，MS-DOSでのメモリ管理ファンクションを書いておこう(図4)．今月のプログラムでもこれを一部使用する．なお，1パラグラフは16バイトである.

図4

```
【メモリ獲得】

呼び出し
          AH = 48H
          BX = 獲得したいメモリサイズ（パラグラフ単位）
戻り値
          CY = 0
                    AX = 獲得したメモリブロックのセグメントアドレス
          CY = 1
                    AX = エラーコード
                    BX = 割り当て可能なメモリサイズ（パラグラフ単位）

【メモリ解放】

呼び出し
          AH = 49H
          ES = 解放したいメモリ領域のセグメントアドレス
戻り値
          CY = 0
                    正常終了
          CY = 1
                    AX = エラーコード

【メモリサイズ変更】

呼び出し
          AH = 4AH
          BX = 新たに指定するメモリサイズ（パラグラフ単位）
          ES = 変更したいメモリ領域のセグメントアドレス
戻り値
          CY = 0
                    正常終了
          CY = 1
                    AX = エラーコード
```

パイプは細いほうがいいというお話

再帰呼び出し処理の中で使われる動的な変数領域にはスタックを使う．そのスタックはCPUスタックではなく，ソフトウェアによって実現し，そのためのメモリ領域はメモリ獲得ファンクションを使ってプログラムが起動されたあとで自分で確保する.

今月のプログラムの重要な位置を占める，再帰呼び出し処理の中での動的な変数領域に関する部分の骨格が決まってきた．もうひといき，最後の詰めをやっておこう.

■モジュール間の結合はできるだけ粗であるほうがよろしい…なんのことだ?

ここで用意しようとしているソフトウェアよるスタックは，1つの独立した機能を果たすモジュールである．独立しているといっても，ファイルが別になっているということを意味するものではない．あくまでも，機能として独立しているということだから誤解のないように.

この独立した機能を果たすモジュールは，いくつかのさらに小さな機能，サブルーチンからなっている．また，このモジュールが管理するデータも含まれている．さて，これらがほかのモジュールから自由にアクセスできるようになっていたら，どういうことが起きるだろうか.

たとえば，ソフトウェアスタックの実現のためにはスタックポインタ（これはソフトウェア的なものだ．CPUのSPレジスタのことではない）を設けなくてはならない．スタックというデータ構造は，このスタックポインタを唯一のよりどころとして動作するのだ．では，このスタックポインタがプログラムのどこからでも自由に参照/変更ができるようになっていたとしたらどうだろう．これはちょっと恐いことになる.

もちろんそれをするのはプログラマの自由だし，その責任において行うことだが，ちょっとした効率アップやプログラムミスのために本来そのモジュールが用意していない機能を，そのモジュールに含まれるデータやローカルなサブルーチンに直接アクセスすることで強引に実現してしまうことがある．

例をあげよう．スタック操作モジュールがスタックの2番目に積まれているデータにアクセスする機能を提供していなかったとする．しかし，外部のモジュールがスタックポインタを直接ずらすことで，これは容易に実現できるのだ．

ところが，本来スタック操作を管理するはずのモジュールはこのことを知らない．こういう不安定な状況で，スタック管理モジュールに正規のスタック操作要求がきたらどうなるだろう．複雑なプログラムでは，そういうことが実に簡単に起こるのだ．こうなってはモジュールの独立もなにもあったものではない．

そういうわけで，独立したモジュールはできるだけ限られた，そして十分管理されたインターフェースで外部とやりとりをすべきであるといえるのだ．で，パイプはできるだけ細いほうがいいというお話になる．たとえ効率は少々落ちても，迷わず安全なやり方を選ぶべきだ．効率のほうは，いずれハードウェアが追いついてくる．

スタック管理モジュールの外部インターフェース

くどいようだが，外部といってもファイルを物理的に外に出してしまうというようなことをいっているのではない．あくまでもモジュールとして見た場合の「外部」なのだ．

スタック操作にはどのような機能が必要か．実はこれ，2つだけだ．PUSHとPOP，これ以外には必要ない．強いてあげるならスタックの初期化がある．これは最初に一回だけ実行すればいい．スタック管理モジュールと外部のモジュールは，この2つの機能でのみ，やり取りをすることにしよう．

このように，外部とのインターフェースを限定するのにいちばん簡単な方法はファイルを分けてしまうことだ．アセンブラでは，あるソースファイルの中で定義されたシンボルは，とくに宣言しなければ外部に対して公開されない仕組みになっているので，いとも簡単に情報の隠蔽ができる．

では，同一ファイルの中でのインターフェースの限定はできないのかというと，けっしてそうではない．ちょっとした命名規則をもち込み，プログラマがそれを遵守することでそれは可能になる．ちょっとした命名規則とは，たとえば直接参照してはいけないシンボルは特定の文字からはじまる名前をもつというようなものだ．アンダースコア2個ではじまっているシンボルは直接参照してはいけない．あるいは，グローバルなシンボルは絶対にアンダースコア2個でははじまらないというような規則を1つ導入するだけで，わざわざソースファイルを分離しなくても情報の隠蔽は可能である．

スタック管理モジュールの構成

では，スタック管理モジュールを設計してみよう．あとで使い回しができるように，柔軟な構造にしておきたいものだ．

スタック管理モジュールを使う立場から指定される要素としては，スタックレベルの最大値（これは要するにスタック用のメモリ領域のサイズのことだ），スタックに積むデータのサイズがある．これを初期化時に指定できるようにしておくと便利だろう．

また，データのやり取りの方法だが，データサイズがたとえばワード値に納まるならばレジスタに直接その値を入れて呼び出せばいいのだが，どのような構造のデータにも対処できるようにするためには，アドレス渡しが便利だ．つまり，PUSHするときにデータのおかれているアドレスを渡すというやり方である．POPするときには，逆にデータを取り出したいバッファのアドレ

図5

```
(1) 初期化

    入力レジスタ
        CX = データサイズ
        AX = スタックの深さ
    出力レジスタ
        AX = 実際に確保したスタックの深さ

(2) PUSH

    入力レジスタ
        DS:SI = データアドレス

(3) POP

    入力レジスタ
        ES:DI = バッファアドレス
```

スを指定すればいい．

ということで，スタック管理モジュールのインターフェースを図5のように決める．

プログラム概説

さて，やっとプログラム概説までたどり着いた．本題に入る前が異様に長いのは歳のせいだから我慢してもらいたい．これでもずいぶんはしょっているのだ．

プログラムのメインロジックは，最初に日本語で書いたとおりだ．なお，ファイルの表示が先か，ディレクトリを探るのが先かという件はどちらでもいいので，こちらの好みでファイルの表示を先にする．ちょっとの改造でディレクトリを探るほうを先にするとこは可能だから，もしその気のある読者はチャレンジしてみてもらいたい．

言葉は悪いが，プログラミングはそうやってサンプルをいじり倒して覚えるものだ．そういう意味では，今月のプログラムは結構いい材料になると思う．少しの変更で，ディレクトリツリーを表示するプログラムや，指定したファイルのあるディレクトリにカレントディレクトリを移動するプログラムを作ることができる．また，発見したファイルのサイズも表示するようにしたり，その合計も計算して表示するようにしたり，工夫しだいでいくらでも楽しめる．指定された文字列をファイル名表示時に先頭や末尾に表示するように改造するのもいいかもしれない．なにに使うのかって？

```
A>T *.BAK 'DEL '>DELBAK.BAT
```

とやったら，全部のバックアップファイルを削除するバッチファイルが簡単に作れるじゃないか．ほかにもいろいろと応用が考えられる．ともあれ，ひととおり読んで理解できたと思ったら，今度は実際にやってみることだ．

おっと，また話が脇道にそれてしまいそうだ．

①メインルーチンは初期化終了後，最初のサブディレクトリ名をもって「サブディレクトリの検索と表示」を呼び出す．
話がややこしくなるので，検索すべきファイル名はグローバル変数においておこう．これは動作中に値が変わらないデータなので，無理矢理スタックだのローカル変数だのに押し込めたり，引数にする必要はない．

②「サブディレクトリの検索と表示」は，まず渡されたサブディレクトリ名の中から検索すべきファイル名を探して表示する．
ワイルドカードが渡されてもいいように，連載の第1回目でやったように「最初のファイルの検索」が成功したら，「次のファイルの検索」を失敗するまで呼び出す．

③次に同じ処理の中で，サブディレクトリを探す．
サブディレクトリが見つかったら，その見つかったサブディレクトリ名を引数にして自分自身を呼び出す（2に飛び込む）．

このような単純なロジックで，指定されたファイルをすべてのサブディレクトリの中から検索することができる．言葉にするとたったこれだけのことで，正直に1つの見逃しもなくすべてのディレクトリを検索してくれるのだから，ソフトウェアというのはほんとうにおもしろい．もっとも，言葉にするとたったこれだけのことを実際のコードにしてちゃんと動かすのはなかなか骨が折れる．なんとかならんかなぁ，このギャップ．

プログラム説明

コマンドラインオプションの解析は，5月号で紹介した“SETPARAM”を使う．第1オプションは検索すべきファイル名で，ワイルドカードの使用も許している．第2オプションは検索を開始するディレクトリである．省略時の値はそれぞれ，“＊.＊”と”¥”である．したがって2つとも省略した場合は，カレントドライブのルートからすべてのファイル名を検索して表示する．

ファイルの再帰的検索を呼び出す場合に渡すサブディレクトリ名は，“g_dta”という構造体の中にセットしておく．この構造体の型紙は“dta”という名前で最初に定義されている．ほんとうはこの型紙は名前のとおりDTAを定義したものなのだが，自分を呼び出したモジュール（これは自分自身かもしれない）から渡されるサブディレクトリ名も退避／復帰の対象になるので，同じ構造体に押し込めてある．どうもMASMのSTRUC命令はネストを許していないらしい．たしかできると思ったのだが，あれはTurbo Assemblerのことであったか．ネストを許しているのならばもっときれいな定義ができたのだが，無念である．リスト1のようにしたかったのだが．

実際の処理を行う“search_and_display”では，まず渡されたディレクトリ名の末尾に検索すべきファイル名（これはグローバル変数“search_name”にセットされている）を追加し，最初のファイルを検索する．検索

するファイルのアトリビュートは，通常ファイルのみである．

発見されたすべてのファイル名の表示が終わったら，今度は同じディレクトリの中のサブディレクトリを検索する．検索するファイルのアトリビュートにサブディレクトリ属性を加えると，通常のファイルのほかにサブディレクトリも検索対象になるので，発見されたファイルのアトリビュートを検査してサブディレクトリのみを対象にする．

サブディレクトリが発見されたら，自分自身に渡されたサブディレクトリ名の末尾にそれを追加して新たなサブディレクトリ名を作成し，それをもって自分自身に飛び込む．

さて，この連載の第1回目で，「次のファイルの検索」は「最初のファイルの検索」で使ったDTAと同じものを使わなくてはならないと書いたことをご記憶だろうか．

自分自身を呼び出して，戻ってきたらその次のサブディレクトリを探さなければならないのでDTAをこのままにしておきたいのだが，飛び込んだ自分自身はまた最初のファイルの検索から処理をはじめるので，そのままではDTAを壊してしまう．

これを避ける方法は2通りあって，1つは呼び出されるごとに新たなDTAを設定する方法．もう1つは，DTAは変更せず，自分自身を呼び出す直前にDTAの内容をいったん退避し，戻ってきたら退避しておいたDTAの内容を復帰してやるという方法である．

今回のプログラムでは，後者の方法をとる．

ソフトウェアで実現したスタックにDTAデータを退避し，戻ってきたら復帰してやるということだ．こうしておけば，DTAアドレスそのものは変更せずに処理が続行できる．

先月号で，動的な変数をスタック上に作るのではなく，普通のレジスタを使っておいて，自分自身を呼び出す直

リスト1

```
dta             struc
        dta_data        db      21  dup (?)
        dta_attr        db      (?)
        dta_time        dw      (?)
        dta_date        dw      (?)
        dta_sizel       dw      (?)
        dta_sizeh       dw      (?)
        dta_name        db      13  dup (?)
dta             ends
;
base_data       struc
        d_data  dta     <>
        p_name          db      80  dup (?)
        p_endp          dw      (?)
base_data       ends
```

リスト2

```
TITLE   Command T.EXE

PAGE    60,128

.xlist
include         console.h
include         dosfunc.h
.list

_TEXT   SEGMENT  WORD PUBLIC 'CODE'
_TEXT   ENDS

_DATA   SEGMENT  WORD PUBLIC 'DATA'
_DATA   ENDS

_STACK  SEGMENT STACK
_STACK  ENDS

DGROUP  GROUP   _DATA,_STACK
        ASSUME DS: DGROUP, SS: DGROUP
;
; ======================== 定数宣言 ===========
;
PSP_PARAM       equu    80h             ; PSPのコマンドラインオプション
                                        ; のオフセット.
ARGMAX          equ     16              ; コマンドラインオプションの最大数.
ARGBUFSIZE      equ     256             ; コマンドラインオプション処理のための
                                        ; バッファのサイズ.
;
STACK_LEVEL     equ     64              ; スタックの深さの最大値.
;
dta             struc                   ; dtaという型紙を宣言
        dta_data        db      21  dup (?)
        dta_attr        db      (?)
        dta_time        dw      (?)
        dta_date        dw      (?)
        dta_sizel       dw      (?)
        dta_sizeh       dw      (?)
        dta_name        db      13  dup (?)
;
        p_name          db      80  dup (?)
        p_endp          dw      (?)
dta             ends
;
; ==================================================
;

_TEXT       SEGMENT
        ASSUME  CS: _TEXT
;
extrn           printmes:near,putchar:near,print_asciz:near
extrn           set_param:near,skip_space:near
;
; =================== ソフトウェアスタックモジュール ===============
;
_DATA   SEGMENT  WORD PUBLIC 'DATA'

__data_size             dw      (?)
__stack_size            dw      (?)
__stack_off             dw      (?)
__stack_seg             dw      (?)

__setuperror_mes        db      'Setup error',0dh,0ah,00h
__pusherror_mes         db      'STACK overflow',0dh,0ah,00h
__poperror_mes          db      'STACK underflow',0dh,0ah,00h

_DATA   ENDS

__stack_error_exit      proc
        call    print_asciz
        mov     ah,DOS_TERMINATE
        mov     al,1
        int     21h
__stack_error_exit      endp

;
; ------------------- スタック初期化
;
; 入力レジスタ  CX = データサイズ
;               AX = 最大スタックレベル
; 出力レジスタ  AX = 実際に確保したスタックレベル
;
setup_stack     proc
        push    bx
```


ワンラインNEWS
日立製作所は3ボルトという低電圧で動作する4Mビット疑似SRAMを開発し，サンプル出荷を開始した．ノート型パソコンなど携帯電子機器で，このSRAMは需要が高い．

```
        push    cx
        push    dx
        mov     __data_size,cx
        mul     cx                      ; 必要なメモリサイズを計算し
        add     ax,15                   ; パラグラフ数に変換して
        jc      __setups_error
        shr     ax,1
        shr     ax,1
        shr     ax,1
        shr     ax,1
        mov     bx,ax
        mov     ah,DOS_MALLOC           ; メモリ獲得する
        int     21h
        jnc     __setups                ; もし獲得できなかったら,
        or      bx,bx                   ; 獲得できそうなパラグラフ数が
        jz      __setups_error          ; ゼロでなければ
        mov     ah,DOS_MALLOC
        int     21h                     ; リトライする.
__setups:
        shl     bx,1                    ; 確保したパラグラフ数を
        shl     bx,1                    ; バイト値に変換したものが
        shl     bx,1
        shl     bx,1
        cmp     bx,__data_size          ; データサイズより
        jb      __setups_error          ; 小さければエラー.
        mov     __stack_off,bx          ; スタックポインタ初期値と
        mov     __stack_size,bx         ; メモリサイズを設定し
        mov     __stack_seg,ax
        mov     ax,bx
        mov     dx,0
        div     [__data_size]           ; 実際に確保したスタックレベルを返す.
        pop     dx
        pop     cx
        pop     bx
        ret
;
__setups_error:
        mov     si,offset __setuperror_mes
        jmp     __stack_error_exit
setup_stack     endp
;
;
; ------------------- P U S H
;
; 入力レジスタ  DS:SI = データアドレス
; 出力レジスタ  なし
;
push_data       proc
        push    cx
        push    si
        push    di
        push    es
;
        les     di,dword ptr __stack_off
        mov     cx,__data_size
        sub     di,cx
        jc      __push_error            ; スタックオーバーフローを検査して
        mov     __stack_off,di          ; オーバーしていなければ
rep     movsb                           ; コピーする.
;
        pop     es
        pop     di
        pop     si
        pop     cx
        ret
;
__push_error:
        mov     si,offset __pusherror_mes
        jmp     __stack_error_exit
push_data       endp

;
; ------------------- P O P
;
; 入力レジスタ  ES:DI = バッファアドレス
; 出力レジスタ  なし
;
pop_data        proc
        push    cx
        push    si
        push    di
        push    ds
;
        mov     si,__stack_off
        cmp     si,__stack_size
        jae     __pop_error             ; スタックアンダーフローを検査して
        mov     cx,__data_size
        mov     ds,__stack_seg
rep     movsb                           ; OKならコピーする.
;
        pop     ds
        mov     __stack_off,si
        pop     di
        pop     si
        pop     cx
        ret
;
__pop_error:
        mov     si,offset __poperror_mes
        jmp     __stack_error_exit
pop_data        endp
;
; ==================================================================
;
;
; ------------------- 最初のファイルの検索
;
; 入力レジスタ  DS:DX=ファイル名先頭アドレス
; 出力レジスタ  AX=エラーコード
;
find_first      proc
        push    bx
        push    cx
;
        mov     ah,DOS_FINDFIRST
        int     21h
;
        pop     cx
        pop     bx
        ret
find_first      endp
;
; ------------------- つぎのファイルの検索
;
; 入力レジスタ  なし
; 出力レジスタ  AX=エラーコード
;
find_next       proc
        mov     ah,DOS_FINDNEXT
        int     21h
        ret
find_next       endp
;
; ------------------- 漢字コード第一バイト目の検査
;
; 入力レジスタ  AL = 検査するコード
; 出力レジスタ  CY = 0:漢字第一バイト
;                  = 1:それ以外
;
iskanji                 proc
        cmp     al,81h
        jc      _isk_exit
        cmp     al,0efh
        cmc
        jc      _isk_exit
        cmp     al,0e0h
        jnc     _isk_exit
        cmp     al,0a0h
        cmc
_isk_exit:
        ret
iskanji                 endp

;
; ------------------- 終端子までコピー
;
; 入力レジスタ  DS:SI = 元データのポインタ
;               ES:DI = コピー先ポインタ
;
strcpy                  proc
        lodsb
        stosb
        cmp     al,STERM
        jnz     strcpy
        ret
strcpy                  endp
;
; ------------------- 発見ファイル名の表示
;
; 入力レジスタ  ES:DI = ディレクトリ名の終端子を指す
;
disp_fname              proc
        push    cx
        push    dx
        push    si
        push    di
;
        mov     si,offset search_name   ; 検索するファイル名を
        push    di
        call    strcpy                  ; ディレクトリ名に追加し
        mov     dx,offset g_dta.p_name
        mov     cx,0                    ; 通常ファイルのみを
        call    find_first              ; 検索する.
        pop     di                      ; 追加したファイル名を
        mov     byte ptr [di],STERM     ; キャンセルしておく.
_dsp_lp:
        jc      _srchf_exit             ; 発見できなければ終わり.
        mov     si,offset g_dta.p_name  ; ディレクトリ名と
        call    print_asciz
        mov     si,offset g_dta.dta_name
        call    print_asciz             ; 発見したファイル名を別々に表示し
        PRINTM  H_STDOUT,<0dh,0ah>      ; 改行して
        call    find_next               ; 次のファイルの検索をする.
        jmp     _dsp_lp
_srchf_exit:
        pop     di
        pop     si
        pop     dx
        pop     cx
        ret
disp_fname              endp
;
; ------------------- ファイルの再帰的検索
;
; 入力レジスタ  なし
;
search_and_display      proc
        mov     di,offset g_dta.p_name
_cp_lp:
        cmp     byte ptr[di],STERM
        jz      _cp_end                 ; ディレクトリ名の最後の文字を探し
        mov     al,[di]
```

ワンラインNEWS
日本モトローラはレーザープリンタなど電子機器制御などに使われる32ビットマイクロコントローラ「MC68330」「MC68EC020」のサンプル出荷を開始した。

```
        call    iskanji
        jc      _cp_asc
        inc     di
_cp_asc:
        inc     di
        jmp     _cp_lp
_cp_end:
        cmp     al,'¥'                  ; '¥'か
        jz      _cp_skp
        cmp     al,':'                  ; ':'か
        jz      _cp_skp
        cmp     di,offset g_dta.p_name  ; 空文字列の場合は
        jz      _cp_skp
        mov     byte ptr[di],'¥'        ; 末尾に'¥'を追加する.
        inc     di
_cp_skp:
        mov     g_dta.p_endp,di
;
        call    disp_fname              ; ファイルを検索して表示し
;
        mov     di,g_dta.p_endp
        mov     si,offset wildname      ; つぎに, 全ファイル名を
        call    strcpy                  ; 検索対象にし
;
        mov     dx,offset g_dta.p_name
        mov     cx,AT_D                 ; ディレクトリ名も検索対象に入れ
        call    find_first              ; 最初のファイルの検索をし
_ddsp_lp:
        jc      _srchd_exit
        test    g_dta.dta_attr,AT_D     ; 発見したファイル名が
        jz      _ddsp_skp               ; ディレクトリ名であり, かつ
;
        mov     si,offset g_dta.dta_name
        cmp     byte ptr[si],'.'        ; ".", ".."でなければ
        jz      _ddsp_skp
;
        mov     di,g_dta.p_endp         ; 渡されたディレクトリ名に追加し
        call    strcpy
;
        mov     si,offset g_dta
        call    push_data               ; ＤＴＡを退避して
        call    search_and_display      ; 自分自身を呼び出し
        mov     di,offset g_dta
        call    pop_data                ; ＤＴＡを復帰して
_ddsp_skp:
        call    find_next               ; 次のディレクトリの検索をする.
        jmp     _ddsp_lp
_srchd_exit:
        ret
search_and_display      endp
;
; ----------------------------------------
;
main    proc
        mov     si,PSP_PARAM + 1        ; DSはPSPを指している.
        mov     ax,DGROUP
        mov     es,ax
        mov     di,offset param
        mov     cx,ARGMAX
        call    set_param               ; 毎度おなじみのパラメータ取得
        mov     ax,es
        mov     ds,ax
;
        push    cx
        mov     cx,size dta             ; データサイズと
        mov     ax,STACK_LEVEL          ; 最大スタックレベルで
        call    setup_stack             ; ソフトウェアスタックを初期化する.
        pop     cx
;
        mov     di,offset search_name
        mov     si,offset wildname      ; デフォルトの検索ファイル名をとり
        cmp     cx,0                    ; 引数がなければそれを
        jz      _mfdef
        mov     si,[param]              ; あればその引数をとり
_mfdef:
        call    strcpy                  ; バッファにコピーして
;
        mov     dx,offset g_dta         ; DTAをセットし
        mov     ah,DOS_SETDTA
        int     21h
;
        mov     di,offset g_dta.p_name
        mov     si,offset default_dir   ; デフォルトの開始ディレクトリをとり
        cmp     cx,2                    ; 引数がなければそれを
        jnz     _mddef
        mov     si,[param + 2]          ; あればその引数をとり
_mddef:
        call    strcpy                  ; バッファにコピーして
;
        call    search_and_display      ; 検索開始.
;
t_end:
        mov     al,NORMAL_END           ; 終了コードを返して終わり.
        mov     ah,DOS_TERMINATE
        int     21h
;
main    endp

_TEXT   ENDS

_STACK  SEGMENT STACK
;
        dw      1024 dup (?)
;
_STACK  ENDS

_DATA   SEGMENT  WORD PUBLIC 'DATA'
;
default_dir     db      '¥',0
wildname        db      '*.*',0
;
search_name     db      80              dup (?)
;
g_dta           dta     <>              ; dta型の構造体を宣言
;
param           dw      ARGMAX          dup (?)
                db      ARGBUFSIZE      dup (?)
;
_DATA   ENDS

        END     main

```

COLUMN　積み上げ方式のサブディレクトリ

この原稿を書きながら，なぜサブディレクトリをうまく作れないのか考えてみたのだが，ひょっとしたらこの構造自体に問題がありはしないだろうか．

われわれが，たとえば机の上やひき出しの中を整理しようとするときには，普段かたっぱしから積み上げたり放り込んでおいたものの中からまとめられるものを選びだして，それを封筒に入れるなりファイルに綴じるなりするだろう．その封筒やファイルがいっぱいたまると，今度はそれをまたひとまとめにしてダンボール箱に入れたりする．

だから，下へ下へとサブディレクトリを掘り下げていくのではなくて，まず適当な場所を作ってそこで作業をする．別の仕事をするときは別の場所を確保する．いろいろとやっているうちに同じ階層にたくさんの作業場所ができてしまうから，それらをざっと眺めてまとまりのつくものを1本にまとめてしまう．というふうに，積み上げ方式のディレクトリ作成ができればもっと楽になるのではないかと考えてみたのだ．

さらに，いったんまとめられたディレクトリの1つをまた独立させることも簡単にできるようになっていれば，仕事の再開が実にスムーズになる．ちょうどダンボールの中の1つの封筒を取り出して，また机の上に広げるようなものだ．

こうしておくと，比較的使用頻度の少ないファイルやディレクトリが下のほうに位置することになるので，大騒ぎをしてファイルを探し回らなくてすむ．逆にいうと使用頻度の高いものがいつも最上位におかれているので，最上位のディレクトリリストがそのまま今取りかかっている仕事の概要あるいは全体像をあらわしていることになる．

もちろん，こういう構造にしても整理の上手下手はでてくるのだが，もともとその人がもっている資質以上のことは，いかなるコンピュータでもひき出すことはできないのだから，問題はあるまい．

前でそのレジスタを退避するという方法を紹介したが，あれと同じことを“g_dta”という仮想的なレジスタとソフトウェアスタックを使って行うということである．

アセンブルとリンク

A> MASM T;

A> LINK T+CONSOLE+SETPARAM,,,,/CP:1

“CONSOLE. H”，“CONSOLE. OBJ”，“SETPARAM. OBJ”は先月までと同じものを使用する．”DOSFUNC. H”は5月号に掲載したものに多少追加したので，リスト3に示すものを使用する．

今月のリンクパラメータには，髭だの尻尾だの妙なものがついているので，少し説明の必要がありそうだ．

今月のプログラムでは，起動後にソフトウェアスタックに必要なメモリをMS-DOSのファンクションを使って獲得する．ところが，このプログラムを普通にリンクすると，これはたいてい失敗する．「そんな馬鹿な，フリーエリアはたくさんあるぞ」といわれるかもしれないが，ここにちょっとした落とし穴があったのだ．

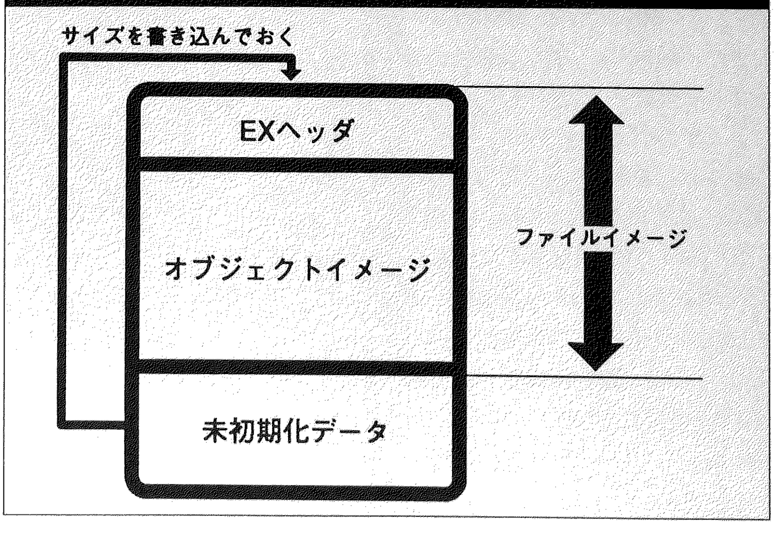
詳しいことを書き出すと長くなってしまうが，EXEタイプのプログラムではオブジェクトファイルの先頭にEXEヘッダというお呪いがつくのだが，この中に「プログラムの実行に必要な最大／最小メモリパラグラフ数」を指定する領域がある．このうち最小メモリパラグラフ数というのは，スタックセグメントも含めたオブジェクトイメージの末尾の未初期化データのサイズことである．未初期化データはその領域だけが確保されればいいので，サイズだけをEXEヘッダに記録しておくことで，オブジェクトファイルのサイズを節約しているのだ（図6）．

これに対して最大メモリパラグラフ数というのは，たとえばヒープメモリ（必要のつどそこから切り出して使うメモリエリア）のような使い方をする領域のサイズで，これはプログラマが書いたロジックに依存する．いかなリンカといえどもプログラマの意図までは理解できないので，やむなくここには考えられる最大の数をセットしておく．つまり0FFFFHだ．これはパラグラフ数であるから，バイト数でいうと1MBにあたる．実際にはフリーエリアがこんなにあるはずはないから，ローダはやむなくそのとき管理しているメモリブロックのうちの最大のものをこのプログラムのために割り当てる（最大メモリパラグラフ数というのは「最大これだけのメモリがほしい」という要求であって，起動時に実際にそれだ

リスト3

```
;;;;;;;;;;;;;;;;;;;;;;
;       dosfunc.h      ;
;;;;;;;;;;;;;;;;;;;;;;
;
H_STDOUT        equ     1                   ; 標準出力ハンドル
H_ERROR         equ     2                   ; 標準エラー出力ハンドル
;
NORMAL_END      equ     0                   ; プロセス正常終了コード
;
AT_A            equ     00100000b           ; アーカイブ属性
AT_D            equ     00010000b           ; サブディレクトリ
AT_V            equ     00001000b           ; ボリュームラベル
AT_S            equ     00000100b           ; システムファイル
AT_H            equ     00000010b           ; 隠しファイル
AT_R            equ     00000001b           ; リードオンリーファイル
;
ATTR_ALL        equ     AT_A + AT_D + AT_V + AT_S + AT_H + AT_R
;
; === ＩＮＴ－２１Ｈ機能番号 ===
;
DOS_SETDTA      equ     1ah                 ; ＤＴＡアドレス設定
DOS_SETVECT     equ     25h                 ; 割り込みベクタ値の設定
DOS_GETVECT     equ     35h                 ; 割り込みベクタ値の取得
DOS_HWRITE      equ     40h                 ; ファイル／デバイスへの出力
DOS_TERMINATE   equ     4ch                 ; プロセス終了
DOS_FINDFIRST   equ     4eh                 ; 最初のファイルの検索
DOS_FINDNEXT    equ     4fh                 ; つぎのファイルの検索
DOS_MALLOC      equ     48h                 ; メモリ獲得
```

図6



けの領域が確保されているという保証はないので注意してほしい).

こういう状況なので，ここでいきなりメモリをよこせといっても，すでにあるだけのメモリが自分自身のために割り当てられているので獲得できないことが多い．獲得できる場合もあるのだが，そのへんの事情についてはまた別の機会に書くことにしたい．

このような状況を回避するための方法は2通りある．1つは起動後にメモリサイズ変更のファンクションを実行して，自分自身に割り当てられたメモリのうち必要なものだけを残して返してしまい，あらためてメモリを獲得にいくという方法．もう1つは，リンカに対して自分が必要な最大メモリパラグラフ数をきちんと教えてやる方法だ．

どちらの方法をとってもいいが，メモリを返すためにはそのぶんのコードを書かなければならないし，どれだけ残せばいいのかという計算はデータセグメントの定義が別れていたりすると意外にめんどうである．かといって必要な最大メモリパラグラフ数といっても，そうおいそれとソロバンをはじくわけにはいかない．今回のようなものなら最小パラグラフ数と同じでいいのだが，その最小パラグラフ数の計算が厄介だ．そこで，ここでも筆者得意の手抜きをする．

リンカに対して必要な最大パラグラフ数を教えるオプションは“/CP”である．正式には“/CPARMAXALLOC”だが，“/CP”で通じる．このあとにコロンに続いてその数値を書いてやればいいのだが，ここに必要な最小メモリパラグラフ数を明らかに下回るサイズを指定してやると，リンカは自動的に最小メモリパラグラフ数と同じものを書き込んでくれる．これでも，実際にメモリを獲得しにいったときにメモリ不足で断られる可能性がないでもないが，その時は実行をあきらめればいいだけだ．

めんどうなようだが，これをちゃんとやっておかないと自分が作ったアセンブラプログラムからはメモリ獲得も外部プログラムの起動もいっさいできないことになる．

試しにリスト4のプログラムをアセンブル，リンクし，カレントディレクトリに“COMMAND.COM”をコピーしておいて実行してみてもらいたい．“CP:1”をつけずにリンクしたものでは“COMMAND.COM”が実行されないことが確認できるだろう．

コンパイラでは，このへんはライブラリの中のスタートアップルーチンがきちんとやってくれているようだが，アセンブラプログラマはすべてを自分の支配下におくことができるかわりに，すべてのことに責任をもたなくてはならない．筆者などこのからくりを知らないで半日ほど悩んだことがあった．

今月のお勧め

今月は，せっかくあるのにめったに使われないSTRUCという命令を使ってみた．C言語を使われる方はstructという機能をご存知だろうが，

リスト4

```
TITLE   Command TEST.EXE

PAGE    60,128

_TEXT   SEGMENT  WORD PUBLIC 'CODE'
_TEXT   ENDS

_DATA   SEGMENT  WORD PUBLIC 'DATA'
_DATA   ENDS

_STACK  SEGMENT STACK
        db      2048 dup (?)
_STACK  ENDS

DGROUP  GROUP   _DATA,_STACK
        ASSUME DS:DGROUP, SS:DGROUP

_TEXT   SEGMENT
        ASSUME  CS:_TEXT
main    proc
        mov     ax,_DATA
        mov     ds,ax
        mov     es,ax
        mov     dx,offset command
        mov     bx,offset param
        mov     ah,4bh
        mov     al,0
        int     21h
        mov     ah,4ch
        int     21h
main    endp
_TEXT   ENDS

_DATA   SEGMENT  WORD PUBLIC 'DATA'

p1              db      6,'/c dir',0dh
;
p2              db      0
                db      15 dup (' ')
;
p3              db      0
                db      15 dup (' ')
;
command         db      'command.com',00h
;
param           dw      0
                dd      p1
                dd      p2
                dd      p3
_DATA   ENDS

        END     main

```

あれと同じようにいくつかのデータをまとめて1つのデータとして扱う機能である．データのタイプは同じでなくてもいい．たとえば，バイト型の変数2個とワード型の変数3個というような組み合わせで，新しいデータ型を定義するわけだ．

これを使うと，いくつかのデータのまとまりをあたかも1つのデータのように扱うことができる．全体に対してのSIZE演算子も適用できるし，逆に個々のデータへのアクセスも自由である．

MASMでの構造体の定義は，構造体の型紙の定義と，それを参照しての実体の定義の2段階に分かれている．型紙というのはデータの構造を定義するものなので，実際にデータエリアが確保されるわけではない．その型紙を参照し，実際のデータエリアを定義して，はじめて参照が許されることになる．

では，わかりやすいようにラベルに日本語を使ってSTRUC命令の使い方を説明しよう（リスト5）．断っておくが，MASMではラベルに日本語は使えないのでこれをこのままアセンブルしてもエラーになる．あくまでも説明をわかりやすくするためなので，その点はご理解いただきたい．

リスト5

```
在庫            struc
        数量    dw      (?)
        単価    dd      (?)
        種別    db      9999
在庫            ends
```

リスト6

```
パソコン在庫    在庫    <>
```

リスト7

```
パソコン在庫    在庫    <1000,198000>
```

リスト8

```
パソコン在庫    在庫    24 dup (<>)
```

これは型紙の定義である．これで「在庫」という型の構造体の型紙が定義されたことになる．「種別」の例のように，型紙の定義のときにデフォルト値を設定することもできる．では，次にこれを参照して実際のデータエリアを確保してみよう（リスト6）．

これは「パソコン在庫」という名前で「在庫」という型のデータエリアを定義したものである．"dw"とか"db"と同じやり方で「在庫」が扱われていることに注目していただきたい．「在庫」というのは"dw"や"db"と同じようなデータ型なのである．

「在庫」の右側の"<>"は，初期値を与えるものである．ここでは初期値が指定されていないので，「数量」と「単価」は"(?)"の状態（実際はゼロになっている）で，「種別」は9999である（この値自体には意味はない）．初期値を与えるにはリスト7のようにする．

これで数量が1000個で，単価が198,000円のパソコンが定義されたことになる．

構造体は，ほかのデータ型と同じように配列としての定義も可能である（リスト8）．

これで「パソコン在庫」という名前で「在庫」というデータ型の要素を24個もつ配列が定義された．

もちろん，アセンブラではこのような機能を使わなくてもデータにアクセスすることは可能だし，このようなめんどうな定義をしなくても同じことを実現することができる．この機能がなかなか使われないのはこのためだろうが，ひとまとまりになって意味を成すデータをそれとわかるように定義しておくというのは，プログラムの見やすさ，理解のしやすさを向上させるために必要なことだ．そういう場面があったらぜひ活用していただきたい．

今月の反省

ちょっと内容が多岐にわたってしまって，書きたりないところがあるかもしれない．いちばん大事な再帰の説明を先月してしまったので今月は安心して書きはじめたのだが，こんな小さなプログラムでも意外に書かなければならないことが多かった．重要なポイントはすべて押さえてあるつもりだが，自分で気がついたところや読者からご指摘のあったところは次回以降で補足するつもりである．

●参考文献

MS-DOSエンサイクロペディア（I）　システム解説編　アスキー出版

98活用アセンブラ技法

ビデオシステム編 第7回

ウィンドウシステムの作成

星野 操

今回は，前々回のbackcol.asm，前回のback2.asmに修正を施したものを利用して，バックカラーをサポートしたウィンドウシステムを作成します．また，ウィンドウシステムの基本機能の実現法についても解説しています．

前回までのリストの修正

diffの出力を利用

まず，前回と前々回のリストについての修正をしておきます．修正リストは，diffの出力を利用します（リスト1）．

前々回の修正は，「code」に関するものです．TASM(Turbo Assembler)はprocの引数などにcodeという文字列を問題なく使用することができますが，MASM（Microsoft Macro Assembler）ではこれが予約語とコンフリクトを起こすのか，エラーになってしまいます．そこで，今回はcodeを単に_codeと置き換えることによって問題を回避しています．この問題は，前回のback2についても発生するので，同様に処理をします．

さらに，前回のリストではsrcoff／srcsegの記述がscroff／scrsegとなっていました．これらはproc内での記述と合致している部分もありましたが，引数にscroffを使用する部分がsrcoffとなっているものもありましたので，全面的にsrcoff／srcsegに修正します．やはり，なるべく省略形は使用しないほうがよいようです．

このミスが発見されなかったのは，TASM 1.0での問題によるものです．TASM 1.0では，引数やローカル変数のマクロのスコープがprocの外側まで及んでしまいます．ですから，前のほうのprocで定義した内容が生き残っていてそれが使用されてしまうのです．

MASMを使った場合は，スコープの管理がきちんとなされているため未定義のラベルを使用したというエラーになります．確認はしていませんが，TASMの後期のバージョンではこの問題は回避されているようです．

最後に，関数「move_from_screen」でBXレジスタに対する更新の手続きが取られていませんでした．これを追加修正します．

diffの出力で「xxx c xxx」とあるのは行内容が変更されたもの，「xxx a xxx」とあるのは追加されたものです．前のxxxは修正前の行番号であり，後ろのxxxは修正後のファイルの行番号です．修正を後ろのほうから行えば，行番号がずれることはありません．

今回のプログラム

ウィンドウシステム

さて，今回は前回作成（今回修正した）したプログラムを利用し，簡単なウィンドウシステムを作ってみます．今回のものでは，アクティブなウィンドウは最後にオープンされたものだけとし，機能はウィンドウのオープン，クローズ，ウィンドウ内のクリア，カーソル移動，文字列出力などに限定しておきます．もちろん，バックカラーはサポートすることにします．

ウィンドウの実現法

これらの機能を実現するために，最初にウィンドウの諸元を記述した構造体を定義し，その配列を作成します．内容としては，まずウィンドウの表示

ワンラインNEWS
日本モトローラは年末をめどに32ビットMPU「68030」を国内で生産する．モトローラはすでに「68040」も発売しているが，「68030」が数量的には主力製品．

域の左上のX，Y座標，そして右下のX，Y座標を用意します．ただし，右下の部分はウィンドウの有効範囲に1をたしたものを使用します．したがって，画面全体を使用する場合には，これらは0，0，80，25となります．1をたしているのは，ウィンドウの桁や行の幅を引き算で簡単に演算できるようにするためです．

そのほかに，ウィンドウ内のカーソル位置(左上部分からのオフセット)，カーソルのスクリーン上のアドレス，ウィンドウを開いたときのスクリーン上の文字の退避バッファのアドレス，枠を使用するかどうかを決定するフラグ，ウィンドウを作成したときにアクティブになっていたウィンドウ記述子へのポインタ，出力する文字のアトリビュートや漢字処理用のバッファなどが用意されています．カーソルのスクリーン上のアドレスを用意しているのは，実際にput_chrなどが使用されるときやカーソルを移動するときなどにはウィンドウ相対ではなく画面上の絶対座標が必要になるからです．

ウィンドウは，枠の使用の可否を決定することができます．枠を使用した場合には，半角スペースで（特定のアトリビュートを用いて）表示域をぐるりと取り囲むことにします．このため，必要な表示範囲は大きくなり，退避バッファなどもより大きなものが必要になります．しかし，当然のことながら見やすくはなります．

退避バッファのアドレスが0である記述子は，フリーな記述子と解釈することにします．したがって，これが記述子の使用フラグになります．

ウィンドウをクローズしたときには，アクティブなウィンドウを切り替えないといけないので，親ウィンドウの記述子を記憶しておきます．また，

リスト1

```
****************************************
* correction of backcol.asm
****************************************

381 c 381
< func      put_chr <mode, adr, code, attr> <di>
----
> func      put_chr <mode, adr, _code, attr> <di>
397 c 397
<                           mov     bx,code
----
>                           mov     bx,_code
401,402 c 401,402
<                           mov     code,ax
<                           or      code,8080h      ; set r/l flag
----
>                           mov     _code,ax
>                           or      _code,8080h     ; set r/l flag
447 c 447
<                   mov     ax,code
----
>                   mov     ax,_code
455 c 455
<                           mov     ax,code
----
>                           mov     ax,_code
476 c 476
<           mov     ax,code
----
>           mov     ax,_code

**************************************
* correction of back2.asm
**************************************

276 a 277
>                           inc     bx
295 a 297,298
>                           inc     bx
>                           shl     bx,1
320 c 323
< func      move_between_screen <scroff, cnt, dstoff> <si di>
----
> func      move_between_screen <srcoff, cnt, dstoff> <si di>
352 c 355
<           calc_ga <adr>
----
>           calc_ga <dstoff>
419 c 422
< func      store_to_screen <mode, adr, cnt, code, attr> <si di>
----
> func      store_to_screen <mode, adr, cnt, _code, attr> <si di>
439 c 442
<           mov     si,code
----
>           mov     si,_code
477 c 480
< func      move_attribute <mode, adr, cnt, scroff, scrseg> <si di>
----
> func      move_attribute <mode, adr, cnt, srcoff, srcseg> <si di>
489,490 c 492,493
<           mov     ds,scrseg
<           mov     si,scroff
----
>           mov     ds,srcseg
>           mov     si,srcoff
538 c 541
< func      move_string <mode, adr, cnt, scroff, scrseg> <si di>
----
> func      move_string <mode, adr, cnt, srcoff, srcseg> <si di>
544,545 c 547,548
<           mov     ds,scrseg
<           mov     si,scroff
----
>           mov     ds,srcseg
>           mov     si,srcoff
638 c 641
< func      set_string <mode, adr, cnt, code> <di>
----
> func      set_string <mode, adr, cnt, _code> <di>
647 c 650
<           mov     ax,code
----
>           mov     ax,_code
671 c 674
< func      write_string <mode, adr, cnt, scroff, scrseg, attr>
----
> func      write_string <mode, adr, cnt, srcoff, srcseg, attr>
```

ウィンドウごとに出力文字のアトリビュートを決定できたほうが便利なので，ウィンドウごとにこれらのためのワークエリアを用意しておきます．

今回は，この記述子のセットを10セット用意しておきましたので，10個までのウィンドウをオープンすることができます．変数wpは，現在アクティブになっているウィンドウの記述子へのポインタを格納するためのものです．

プログラムの解説

back3.asm

get_corner

get_cornerはウィンドウの左上・右下の座標をレジスタに返すものです．ただし先ほど述べたように，右下の値はそれぞれ1をたしたものとなっています．モード引数は，0～2の値を取り，0の場合は表示域，1の場合はウィンドウ域，2の場合は左右マージンを含めた座標を返します．

1の場合，もし枠が使用されているモードであれば，周りに1キャラクタの枠が描かれているわけですから，左上の座標は1を引かなければいけませんし，右下の座標は1をたさないといけません．枠が使用されていなければ，表示域の値と同じになります．

2のモードは，必要なら左右にさらに1キャラクタ分のマージンを取った値を返します．単にウィンドウをオーバーライトした場合，それが漢字を半分に切断する形になると漢字がはみだすなどの不都合が生じます．よって，左端が漢字2バイト目にかかった場合，また右端で漢字の1バイト目にかかった場合は，それぞれ隣の位置のキャラクタをクリアしなければなりません．このために，マージンが必要になるのです．また，この理由によって，ウィンドウ左端・右端が画面の左端・右端と合致するときにはマージンが必要なくなります．この関数では，これらの処理を行っています．

set_corner

次のset_cornerは，端点の座標を格納するものです．右下の座標については，前記のように1をたしてから設定しています．

calc_buf_size

calc_buf_sizeは，退避に必要なバッファのサイズを取得するためのものです．実際には，座標それ自体を引数に取ったほうが便利だと思うのですが，ここでは単にウィンドウ記述子へのポインタを引数に取っています．したがって，適当な記述子に座標を設定してから呼び出すことになります．

calc_buf_sizeは，まずwpの内容を取り替えてからget_cornerで端点の座標を取得し，それらを引くことによってウィンドウ幅を計算し，必要なバッファサイズを取得しています．バッファへの画面コードの退避は，復元性のためにワードモードで行いますので，1キャラクタについて4バイトの領域が必要です．計算された値はAXレジスタを用いて返されます．

calc_lu_address

calc_lu_addressは，画面左上の位置の画面上の物理アドレスを計算するもので，やはり同様なモードの引数を取ります．これは，画面内のオフセットを画面上の物理アドレスに変換する際などに使用します．

calc_width

calc_widthは，行・桁のウィンドウ幅を計算するものです．右下の座標は調整してありますから，単に引き算す

リスト2

```
        .data
window_dp       struc
x_lu    db      ?                       ; left upper corner x poisition
y_lu    db      ?                       ; y position
x_rl    db      ?                       ; right lower corner x position+1
y_rl    db      ?                       ; y position+1
x_c     db      ?                       ; cursor x position
y_c     db      ?                       ; cursor y position
pos_c   dw      ?                       ; cursor video address
buf_ptr dd      ?                       ; buffer pointer
frame   db      ?                       ; frame flag
mother  dw      ?                       ; mother window
c_attr  dw      ?                       ; attribute type
c_code  dw      ?                       ; full size char 1st byte code
window_dp       ends

window_number   =       10

window_dp_array window_dp       window_number dup (<0,0,0,0,0,0,0,0>)
window_dp_end   label   byte

wp      dw      ?                       ; window descriptor pointer
        .code

;------------------------------
; ウィンドウの座標をレジスタに返す
;
; get_corner: get corner address into register
; argument:
;       mode: 0 for display space, 1 for whole window, 2 with margin
; return:
;       AH:y lu position of window
;       AL:x lu position of window
;       DH:y rl position +1 of window
;       DL:x rl position +1 of window

func    get_corner      <mode>
        ; get pointer
        mov     bx,wp
        mov     al,[bx].x_lu
        mov     ah,[bx].y_lu
```

るだけで幅を計算できます.

clear_window

clear_windowは，ウィンドウ内の表示領域のみをクリアするものです．まずcalc_lu_addressで物理アドレスを計算し，それからcalc_widthで幅を計算します．クリアした部分のアトリビュートは表示アトリビュートを流用するので，これをローカル変数にコピーしておきます．それから，行幅の回数，桁幅のキャラクタだけスペースを埋め込みます．このためにstore_to_screenを呼び出します.最後にポインタを更新し,ループを形成しています.

restore_window

restore_windowは，退避されたバッファの内容を画面上に復元するものです．対象となる領域は，現在のwpで示される記述子に記述されている内容となります．一応バッファエリアのアドレスをチェックし，これがクリアされていた場合にはAX=－1としてリターンすることにしています．

それからclear_windowと同様にアドレスや幅の計算をし，ローカル変数にセットします．clear_windowと違うのは，store_to_screenのかわりにmove_to_screenを使用していることだけです．この場合はワードモードを使用するので，第1引数がclear_windowとは異なり1になっています．バッファソースアドレスの更新は，windowの桁幅の4倍単位で行っています．

save_window

save_windowは，restore_windowで使用できるように指定された画面上の領域をセーブするものです．内容はほぼrestore_windowと同じですが，呼び出される手続きがmove_to

```
          mov     dl,[bx].x_rl
          mov     dh,[bx].y_rl

          ; check range
          cmp     mode,0
          @if     nz
                  ; check frame margin
                  cmp     [bx].frame,0
                  @if     nz
                          dec     al
                          dec     ah
                          inc     dl
                          inc     dh
                  @endif

                  cmp     mode,2
                  @if     z
                          ; check r/l margin
                          or      al,al
                          @if     nz
                                  dec     al
                          @endif

                          cmp     dl,80
                          @if     b
                                  inc     dl
                          @endif
                  @endif
          @endif
          ret
  endf


  ;------------------------------
  ; ウィンドウの座標を格納する
  ; set_corner: set view area corner address
  ; argument:
  ;       x1, y1, x2, y2: view port corner position.
  ; return:
  ;       none

  func    set_corner      <x1,y1,x2,y2>
          mov     bx,wp
          mov     ax,x1
          mov     [bx].x_lu,al
          mov     ax,y1
          mov     [bx].y_lu,al
          mov     ax,x2
          inc     ax
          mov     [bx].x_rl,al
          mov     ax,y2
          inc     ax
          mov     [bx].y_rl,al
          ret
  endf


  ;------------------------------
  ; バッファサイズを取得
  ; calc_buf_size: calculate window saving buffer size
  ; argument:
  ;       wp: window descriptor pointer
  ; return:
  ;       AX: buffer size

  func    cacl_buf_size   <wpx>
          push    wp
          mov     ax,wpx
          mov     wp,ax

          get_corner <2>
          sub     dh,ah                   ; get character number
          sub     dl,al
          mov     al,dh
          mul     dl

          add     ax,ax                   ; buf unit size is 4
          add     ax,ax
          pop     wp
          ret
  endf


  ;------------------------------
  ; 画面上の物理アドレスを計算
  ; calc_lu_address: calculate screen address of left upper window corner
  ; argument:
  ;       mode: 0 for display space, 1 for full window, 2 for with margin
  ; return:
  ;       ax: calclated address

  func    calc_lu_address <mode>
          get_corner <mode>
```

_screenからmove_from_screenとなっています．バッファアドレスのチェックは行っていません．

ここでセーブ・リストアされる領域は，左右端のマージンを含めたものとなっています．これは，これらの部分の漢字コードが特別に処理されるかもしれないからです．

set_window_edge

set_window_edgeは，左右マージン桁の処理をするものです．この関数は，簡単にするために他の関数のようにback2. asmの関数を呼び出すことをせず，直接画面を触っています．

同様に各種ウィンドウのパラメータを計算したあと，左右マージン処理が必要であれば，上から下まで漢字を切っていないかをチェックし，切っている場合には漢字の他端を空白に置き換える処理をします．漢字であるかどうかは，コードの上位バイトが0であるかどうかを調べればわかり，右半分であるか左半分であるかは8080Hでマスクを取ってみればわかります．

cursor

cursorは，ウィンドウ内のカーソル位置を制御するための関数です．基本的には，引数に取った値をwpで示される記述子の中に設定するだけですが，さらに漢字処理用のコードをクリアしています．カーソル移動が行われたら，1バイト目は無効扱いにします．

また，画面上のアドレスを計算してそれをpos_cに代入し，その値を利用してカーソルの物理的な移動も行っています．このためにcursorはbiosを呼び出しています．

物理アドレス計算にテーブリングの技術を利用すると，より高速に行えますが，ここでは単にmulを使用してい

```

        ; calculate
        sub     dx,dx
        mov     dl,al
        add     dx,dx
        xchg    ah,al
        mov     ah,160
        mul     ah
        add     ax,dx

        ret
endf


;---------------------------
; ウィンドウの幅を計算
; calc_width: calclate other parameters
; argument:
;       mode: 0 for display space, 1 for full window, 2 for with margin
; return:
;       al: column width
;       ah: row width

func    calc_width      <mode>
        get_corner <mode>

        xchg    ax,dx
        sub     ah,dh
        sub     al,dl
        ret
endf


;---------------------------
; ウィンドウ内をクリア
; clear_window: clear window area
; argument:
;       none
; return:
;       none

func    clear_window    <>,<si di>
        local attr

        calc_lu_address <0>
        mov     si,ax                   ; si for pointer
        calc_width <0>
        mov     di,ax                   ; di for works
        mov     bx,wp
        mov     ax,[bx].c_attr
        mov     attr,ax

        @begin
                mov     ax,di
                or      ah,ah           ; check row count
        @while nz
                mov     ah,0            ; clear
                store_to_screen <0,si,ax,' ',attr>

                add     si,160
                xchg    ax,di
                dec     ah
                xchg    ax,di
        @repeat

        ret
endf


;---------------------------
; バッファの内容を画面上に復元
; restore_window: restore contents of buffer, and close window
; argument:
;       none
; return:
;       AX: 0 for OK, -1 for ERROR

func    restore_window  <>,<si di>
        local   screen_address,window_width,row_count

        mov     di,wp
        les     si,[di].buf_ptr
        mov     di,es
        mov     ax,di
        or      ax,si
        @if     z
                ; no buffer reserved, inactive window
                mov     ax,-1
                ret
        @endif

        ; calculate local works
        calc_lu_address <2>
        mov     screen_address,ax
        calc_width <2>
```


ワンラインNEWS
日本テキサス・インスツルメンツは特定分野向けIC分野を強化するために製品設計を行うデザインセンターを横浜営業所内に置く．

ます．したがって，あまりひんぱんにcursorを呼び出すのは必ずしも適当ではありません．

このため，とくに水平移動をするためにhcursorという別の関数を用意しておきます．これは水平の移動キャラクタ数（正負）を引数に取り，それを用いてカーソルのX座標および画面上の物理アドレスを更新し，カーソル移動のbiosコールを行います．もちろん，移動後のカーソル位置がウィンドウ内にあることが保証されなければなりません．この場合，移動後のアドレスなどの計算は単なるたし算のみで済み，かけ算が必要ないので処理が高速になることが期待されます．ウィンドウに1文字出力などを行う場合に，この関数を呼び出せばよいわけです．

set_color

set_colorは，文字出力のためのアトリビュートをウィンドウ記述子の中に設定するためのものです．引数に取るアトリビュートは，バイトタイプのものです．これは，文字出力を行うときにバイトタイプの書き込みを使用するためです．ワードタイプのアトリビュートを設定したときには，正常に動作しないので注意しなければいけません．

set_frame

set_frameは，ウィンドウがオープンされた場合にフレームを描くものです．連載の最初に記載していた半角外字を用いたダブルボーダーなどを利用できれば，見映えがよくなりますが，今回はこれは使用せず，単に半角スペースを特別なアトリビュートを用いて描くことによって枠を構成しています．アトリビュートは，引数で指定します．

この場合も他の関数と同様に各種パラメータを計算しています（ただし，モ

```
          sub     cx,cx
          xchg    cl,ah
          mov     window_width,ax
          mov     row_count,cx

          ; loop
          @begin
                  cmp     row_count,0
          @while  nz
                  move_to_screen  <1,screen_address,window_width,si,di>

                  mov     ax,window_width ; update pointers
                  add     ax,ax
                  add     ax,ax
                  add     si,ax
                  add     screen_address,160
                  dec     row_count
          @repeat

          sub     ax,ax
          ret
  endf



  ;-----------------------------
  ; 画面上の領域をセーブ
  ; save_window: save window contents to buffer
  ; argument:
  ;       mode: 0 for display space, 1 for whole window, 2 for with margin
  ;       dstoff: distination buffer offset
  ;       dstseg: distination buffer segment
  ; return:
  ;       none

  func    save_window     <mode,dstoff,dstseg>
          local   screen_address,window_width,row_count

          ; calculate local works
          calc_lu_address <mode>
          mov     screen_address,ax
          calc_width <mode>
          sub     cx,cx
          xchg    cl,ah
          mov     window_width,ax
          mov     row_count,cx

          ; loop
          @begin
                  cmp     row_count,0
          @while  nz
                  move_from_screen  <1,screen_address,window_width,dstoff,dstseg>

                  mov     ax,window_width ; update pointers
                  add     ax,ax
                  add     ax,ax
                  add     dstoff,ax
                  add     screen_address,160
                  dec     row_count
          @repeat

          sub     ax,ax
          ret
  endf


  ;-----------------------------
  ; 左右マージン桁の処理
  ; set_window_edge: clear window edge character
  ; argument:
  ;       none
  ; return:
  ;       none

  func    set_window_edge <>
          local   row_width,col_width,screen_address

          calc_lu_address <2>
          mov     screen_address,ax
          calc_width <2>
          sub     dx,dx
          xchg    dl,ah
          mov     row_width,dx
          mov     col_width,ax

          ; set segment
          mov     ax,TEXT_SEG
          mov     es,ax

          get_corner <1>
          or      al,al
          @if     nz
                  ; need to clear
                  mov     bx,screen_address
```

ード＝1で)が，1つ異なるのは，frame_offsetに桁幅−1をセットしていることです．これは，「植木算」の世界で，実際に枠を描き込むべきアドレスを計算するためにはこのframe_offsetの値が存在したほうが有利なのです．

関数は，まず最初に上端の枠をstore_to_screenで描いてから，枠の左端・右端を順次描き，最後に下端の枠を描きます．ここで，左端位置は最初に計算したscreen_addressから順次160をたしていけば求められますが，右端位置はこれにframe_offsetをたしたものとして計算しています．

open_window

open_windowは，これまでに述べてきた各種関数を呼び出してウィンドウを開くものです．

引数はかなり多く，表示域の左上・右下の座標，また枠フラグと画面上のデータの退避バッファアドレス，初期の表示アトリビュートと枠のアトリビュート（いずれもバイトモード）を取ります．

最初に，空いているウィンドウ記述子を探します．記述子の先頭（実際には，さらに要素のサイズを引いたもの）にポインタを設定し，順次退避バッファのアドレスを探り，空いているものがあればそれを使います．見つからなければ，AX＝0（NULL）として返ります．ここでは，size演算子を利用して要素サイズを取得しています．

さて，空いている記述子が見つかったら，そのmotherに現在のwpの内容を設定し，wpの内容を更新します．これ以降，この見つかった記述子が処理の対象となります．まず，表示域の座標をset_cornerを呼び出すことによって設定し，フレーム使用フラグをワークエリアに設定します．また，c

```
                mov     cx,row_width
                @begin
                        ; get code
                        mov     ax,es:[bx]
                        @mc
                                or      ah,ah           ; full width code
                        @cond   nz
                                test    ax,8080h        ; left half
                        @cond   z
                        @if
                                mov     word ptr es:[bx],' '
                        @endif

                        ; update pointer
                        add     bx,160
                        dec     cx
                @until  z
        @endif

        cmp     dl,80
        @if     b
                ; need to clear
                mov     bx,screen_address
                add     bx,col_width
                add     bx,col_width
                dec     bx
                dec     bx
                mov     cx,row_width
                @begin
                        ; get code
                        mov     ax,es:[bx]
                        @mc
                                or      ah,ah
                        @cond   nz                      ; full width code
                                test    ax,8080h
                        @cond   nz                      ; right half
                        @if
                                mov     word ptr es:[bx],' '
                        @endif

                        ; update pointer
                        add     bx,160
                        dec     cx
                @until  z
        @endif

        ret
endf


;----------------------------
; カーソル位置の制御
; cursor: move cursor
; argument:
;       x, y: position in the view area
; return:
;       none

func    cursor  <x,y>
        ; store value
        mov     bx,wp
        mov     cx,x
        mov     [bx].x_c,cl
        mov     ax,y
        mov     [bx].y_c,al

        mov     [bx].c_code,0

        ; add offsets
        add     cl,[bx].x_lu
        add     al,[bx].y_lu

        ; call bios
        mov     dx,160
        mul     dx
        add     ax,cx
        add     ax,cx
        mov     [bx].pos_c,ax
        mov     dx,ax
        mov     ah,13h
        int     18h

        ret
endf


;----------------------------
; カーソルの水平移動処理
; hcursor: horizontal cursor movement
; argument:
;       d_x: character offset
; return:
;       none
```


ワンラインNEWS
富士通は32ビットデスクトップ型パソコン「FMR-50HE I/FE I」など3機種を発売した．ICメモリカードや3.5インチ光磁気ディスクへの対応を特徴としている．

_codeをクリアしておきます.表示アトリビュートは,引数を直接set_colorの引数に取ることによって設定します.

また，退避バッファのアドレスを設定し，save_windowでウィンドウに隠される部分の画面を退避します．そして，set_window_edgeで左右端の漢字処理を行い，必要に応じてset_frameを呼び出してフレームの設定をします．それからclear_windowで表示領域をクリアし，cursorを呼び出すことによってカーソルをホームポジションに設定します.

最後に，記述子へのポインタをAXに返して終了します.この返り値は,トップのウィンドウだけをアクティブにするという決まりを守っているかぎり必要ありませんが,「オーバーラップしない」ウィンドウを切り替えて使用する場合（そのための関数を追加する必要がありますが)には,この返り値を引数に取って切り替えることになります.

close_window

open_windowで開いたウィンドウは，close_windowで閉じることになります．この場合，閉じるウィンドウはwpで示されるので,とくに引数はありません．最初に退避バッファの内容をチェックします．それからrestore_windowで退避した内容を復元し,バッファポインタの内容をクリアします．motherの内容をwpに設定してアクティブなウィンドウを切り替えたあと，それが有効なものであればカーソル位置の設定を行います．ここで，カーソル位置のワークエリアはバイトサイズなので，一度レジスタに読み込んでからcursorの引数にしています.

スクロール機能

画面操作としては，このほかにスク

```
func    hcursor <d_x>
        mov     ax,d_x
        mov     bx,wp
        add     [bx].x_c,al
        add     ax,ax
        add     ax,[bx].pos_c
        mov     [bx].pos_c,ax
        mov     dx,ax
        mov     ah,13h
        int     18h

        ret
endf


;---------------------------
; 文字出力アトリビュートの設定
; set_color: set string attribute
; argument:
;       attr: attribute to set (must be byte type)
; return:
;       AX: old attribute

func    set_color       <attr>
        mov     bx,wp
        mov     ax,attr
        mov     [bx].c_attr,ax
        ret
endf


;---------------------------
; フレームを描く
; set_frame: write frame
; argument:
;       attr: attribute of frame (byte mode)
; return:
;       none

func    set_frame       <attr>
        local   col_width,row_width,frame_offset,screen_address

        ; get parameters
        calc_width <1>
        sub     cx,cx
        xchg    cl,ah
        mov     col_width,ax
        mov     row_width,cx
        dec     ax
        add     ax,ax
        mov     frame_offset,ax

        calc_lu_address <1>
        mov     screen_address,ax

        ; write upper frame
        store_to_screen <0,screen_address,col_width,' ',attr>
        add     screen_address,160

        @begin
                cmp     row_width,2
        @while  a
                store_to_screen <0,screen_address,1,' ',attr>
                mov     ax,screen_address
                add     ax,frame_offset
                store_to_screen <0,ax,1,' ',attr>
                add     screen_address,160
                dec     row_width
        @repeat

        store_to_screen <0,screen_address,col_width,' ',attr>

        ret
endf


;---------------------------
; ウィンドウを開く
; open_window: open window area
; argument:
;       x1,y1,x2,y2: position of corner of display area
;       frame_flag: frame flag
;       bufoff,bufseg: buffer address
;       attr: filling attribute
;       frame_attr: attribute of frame
; return:
;       AX: window descriptor, 0 for error

func    open_window     <x1,y1,x2,y2,frame_flag,bufoff,bufseg,attr,frame_attr>,<si>
        ; search window descriptor
        mov     si,offset window_dp_array-size window_dp
        @begin
                add     si,size window_dp
                cmp     si,offset window_dp_end
```

ロール機能が必要になります．上下・左右スクロールを任意領域に対して行うことができるようにしておけば，いちばん汎用性が高いのですが，ここでは文字出力に必要最低限の画面全体を対象にした垂直スクロールだけをインプリメントすることにしました．ただし，スクロール行数は指定可能とし，逆スクロールもサポートしています．またIBM PCのVIDEO BIOSのように，スクロール行数が0であればクリアを実行することにしています．

まず最初にスクロール行数のチェックを行い，これが0であればclear_windowを呼び，クリアを実行します．次に，お決まりの画面上のアドレスとウィンドウの幅を計算します．表示用のアトリビュートもローカルワークにコピーしておきます．スクロール幅がウィンドウ幅より大きければ，この場合もクリアを呼び出して画面消去をします．diには，ウィンドウ幅－スクロールライン数，つまり実際にコピーをする行数を設定します．

次に，画面上のソースとディスティネーションのオフセット値を計算します．これは引数lineに160をかければよいのですが，正負の処理をするのでimulを利用します．これをsource_offsetに格納し，結果を8000Hと比較することで順方向のスクロールか逆方向のスクロールかを判別します．

順方向であれば，diの回数だけ画面内のコピーを実行します．これが終了したら，スクロールインした行数，つまりlineの数だけ画面クリアを実行します．このとき引数に取るアトリビュートは，ウィンドウの表示アトリビュートです．

逆スクロールの場合もほぼ同様の処理になりますが，コピーを順方向に行うことができないので，ポインタの補

```
                @if     nc
                        ; not found error
                        sub     ax,ax
                        stc
                        ret
                @endif

                ; buffer area check
                les     bx,[si].buf_ptr
                mov     ax,es
                or      ax,bx
        @until  z

        ; set parameter
        mov     ax,wp
        mov     [si].mother,ax
        mov     wp,si

        set_corner <x1,y1,x2,y2>
        mov     ax,frame_flag
        mov     [si].frame,al
        mov     [si].c_code,0
        set_color <attr>

        mov     ax,bufoff
        mov     word ptr [si].buf_ptr,ax
        mov     bx,bufseg
        mov     word ptr [si].buf_ptr+2,bx

        save_window <2,ax,bx>
        set_window_edge <>
        cmp     frame_flag,0
        @if     nz
                set_frame <frame_attr>
        @endif
        clear_window <>
        cursor <0,0>

        mov     ax,si
        clc
        ret
endf


;-----------------------------
; ウィンドウを閉じる
; close_window: close opened window
; argument:
;       none
; return:
;       AX: 0 for OK, -1 for ERROR

func    close_window    <>
        ; check buf pointer
        mov     bx,wp
        les     dx,[bx].buf_ptr
        mov     ax,es
        or      ax,dx
        @if     z
                ; error condition
                mov     ax,-1
                ret
        @endif

        ; restore contents
        restore_window <>

        ; clear buf pointer
        mov     bx,wp
        mov     word ptr [bx].buf_ptr,0
        mov     word ptr [bx].buf_ptr+2,0
        mov     bx,[bx].mother
        mov     wp,bx
        or      bx,bx
        @if     nz
                sub     ax,ax
                sub     dx,dx
                mov     al,[bx].x_c
                mov     dl,[bx].y_c
                cursor <ax,dx>
        @endif

        ; return
        sub     ax,ax
        ret
endf


;-----------------------------
; スクロール処理
; vscroll: vertical scroll
; arguemnt:
;       line: scroll line
; return:
;       none
```

正を行い，画面の下端からコピーをはじめることになります．ポインタの更新も，負の方向に行うのです．

ウィンドウ内表示ルーチン

さて，最後にこのようにして構成したウィンドウの中への表示ルーチンです．前回のback2では，画面アクセスが高速になるように似て非なるルーチンを多く用意しました．呼び出し側で画面の端やそのほかの処理をできる場合には，これらの高速ルーチンを使用することができるのですが，今回のウィンドウのようにそれらの処理が比較的めんどうである場合には，やはり１文字単位の表示をするほうが簡単です．

そこで，１文字表示ルーチンprint_charと，文字列を文字に分解してprint_charを呼び出すprint_stringルーチンを用意することにしました．

print_charでは，ウィンドウの右端にかかる漢字の処理を考えなければいけません．これがなければ，put_chrを呼び出すだけで処理が行えるのですが…．

そこで，sjisコードの１バイト目がきた場合には，それを内部ワークに保存し，２バイト目がきたときにウィンドウ右端にカーソルがあれば，スペースを１文字出力してウィンドウ左端にカーソルを移動させることにします．

まず，文字コードをチェックし，コントロールコードの処理を最初に済ませます．CRコードであれば，x_cのマイナスの値を引数に取ってhcursorを呼び出すことで高速にワークエリアとカーソル位置の更新をします．LFコードであれば，まずスクロールが発生するかどうかをチェックし，発生すればvscrollを呼び出し，発生しなければカーソル位置を下の行に移します．このとき，pos_cの値の更新も必要です．カ

```
func	vscroll	<line>,<si di>
	local	col_width,row_width,source_offset,attr

	; check scroll line
	cmp	line,0
	@if	z
		; zero line for clear
		clear_window	<>
		ret
	@endif

	; get position
	calc_lu_address <0>
	mov	si,ax			; distination pointer

	calc_width <0>
	sub	dx,dx
	xchg	ah,dl
	mov	col_width,ax		; column width
	mov	row_width,dx
	sub	dx,line
	mov	di,dx			; actual scrolling line

	mov	bx,wp
	mov	bx,[bx].c_attr
	mov	attr,bx

	; check area
	@if	be
		; too small view area
		clear_window	<>
		ret
	@endif

	; check scroll direction
	mov	ax,160			; calc source pointer offset
	imul	line
	mov	source_offset,ax

	cmp	ax,8000h
	@if	b
		; plus offset, scroll up
		@begin
			mov	ax,si
			add	ax,source_offset
			move_between_screen <ax,col_width,si>

			; update pointer
			add	si,160
			dec	di
		@until	z

		; clear bottom area
		mov	di,line
		@begin
			store_to_screen <0,si,col_width,' ',attr>
			add	si,160
			dec	di
		@until	z

	@else
		; minus offset, scroll down
		; adjust pointer
		mov	ax,row_width
		dec	ax
		mov	dx,160
		mul	dx
		add	si,ax

		@begin
			mov	ax,si
			add	ax,source_offset
			move_between_screen <ax,col_width,si>

			; update pointer
			sub	si,160
			dec	di
		@until	z

		; clear top area
		mov	di,line
		@begin
			store_to_screen <0,si,col_width,' ',attr>
			sub	si,160
			dec	di
		@until	z
	@endif

	ret
endf
```

ワンラインNEWS
日立製作所は福岡市の九州支店内に業務ソフトの開発部門を開設した．首都圏での人材不足に対応し，九州の優秀な人材を確保するのが狙い．

ーソルの移動は，引数0でhcursorを呼び出すことによって行います．

BSコードであれば，ウィンドウ左端にカーソルがないかぎりhcursorを利用してカーソルを1文字左に寄せます．簡単にするために，カーソルの行ラップアラウンドはしないようにしてあります．PC-9801のビデオドライバでは，この操作を行うようになっていますが，IBM PCのドライバではBSコードで行頭から行末にカーソルを移動することはできません．

TABの場合は，カーソルのX位置が8で割り切れるまでspace出力をします．再帰を利用して簡単にしています．ただし，スペース処理だけは通常文字の中でも特別扱いをしています．

さて，そのほかの文字はすべて通常文字として扱います．まず前記の理由でスペースを特別扱いし，これがきたらput_chrでカーソル位置にスペースを書き込みます．そうでなければ，すでに漢字の1バイト目が送られてきているかどうかをチェックします．現在の値が漢字2バイト目であれば，ウィンドウの右端チェックをし，右端にかかっていればスペースを再帰で出力します．それから，put_chrを利用して1バイト目を出力し，hcursorでカーソル位置を更新した後，pur_chrで2バイト目も出力します．

漢字1バイト目が送られてきていなければ，現在の文字コードが漢字1バイト目であるかどうかをput_chrと同様なアルゴリズムで（簡略に）チェックします．もしそうであれば，ワークエリアに値を設定してすぐ返ります．以上の条件がいずれも満たされなければ，単なる半角文字が送られてきているので，これをput_chrを利用して画面に書き込みます．

最後に，カーソル位置の更新をしま

```
;------------------------------
; 1文字出力
; print_char: print one character
; argument:
;         chr: character to print
; return:
;         none

func    print_char <chr>,<si>
        ; check buffer

        mov     ax,chr
        mov     si,wp

        cmp     al,0dh
        @if     z
                ; cr code
                mov     al,[si].x_c
                mov     ah,0
                neg     ax
                hcursor <ax>
                ret
        @endif

        cmp     al,0ah
        @if     z
                ; lf code
                mov     al,[si].y_c
                inc     al
                mov     ah,al
                add     al,[si].y_lu
                cmp     al,[si].y_rl
                @if     ae
                        ; roll up
                        vscroll <1>
                @else
                        add     [si].pos_c,160
                        mov     [si].y_c,ah
                @endif
                hcursor <0>
                ret
        @endif

        cmp     al,08h
        @if     z
                ; bs code
                mov     al,[si].x_c
                or      al,al
                @if     nz
                        hcursor <-1>
                @endif
                ret
        @endif

        cmp     al,09h
        @if     z
                ; tab code
                @begin
                        print_char <' '>
                        mov     al,[si].x_c
                        and     al,7
                @until  z
                ret
        @endif

        cmp     al,20h
        @if     z
                put_chr <0,[si].pos_c,' ',[si].c_attr>
        @else
        cmp     [si].c_code,0
        @e_if   nz
                ; full size code, 2nd char
                ; check right edge
                mov     al,[si].x_c
                inc     al
                add     al,[si].x_lu
                sub     al,[si].x_rl
                @if     ae
                        ; dummy space output
                        print_char <' '>
                @endif

                ; output code
                put_chr <0,[si].pos_c,[si].c_code,[si].c_attr>
                hcursor <1>
                put_chr <0,[si].pos_c,chr,[si].c_attr>
                mov     [si].c_code,0
        @else
        rol     al,1
        add     al,0c0h
        not     al
        test    al,81h
        @e_if   z
                ; first character
```

す．文字出力後のカーソルが右端にきた場合には，再帰呼び出しでCR／LFを出力してカーソル位置を更新し，そうでなければ，単にhcursorを呼び出すことによって更新を実行します．

print_stringは単に文字列を分解し，0文字までprint_charを呼び出すだけです．ここではショートポインタを利用していますが，常駐ライブラリなどとして利用する場合には，セグメントも引数に取り，長いポインタとしてアクセスしたほうがよいでしょう．

以上でシステムの説明は終りますが，このほかにも部分スクロールや左右スクロール，これに伴う部分クリア，オーバーラップしないウィンドウの切り替えのサポートなどを追加するとさらに使いやすくなるでしょう．また，アトリビュートのみの更新機能もつけ加えると便利かもしれません．

サンプルプログラム

entry

最後に，これらを利用する極めて簡単なサンプルをentry以降に示します．データ領域にはウィンドウの退避領域としてwbuf，出力文字列としてdstrを用意します．main以降では，まずopen_windowを利用してフレームつきのウィンドウをオープンし，さらにキー入力があるまでHello worldをウィンドウ内で出力し続けます．そして，キー入力があればclose_windowでウィンドウをクローズし，disable_cgでCGの設定を解除して終了します．

前々回のbackcol.asm，前回のback2.asmを修正したもののあとに，今回のback3.asmを結合し，アセンブル・リンクすることによって実行可能なプログラムが生成されます．

```
                        mov     ax,chr
                        mov     [si].c_code,ax
                        ret
                @else
                        ; normal half width character
                        put_chr <0,[si].pos_c,chr,[si].c_attr>
                @endif

                ; cursor move
                mov     al,[si].x_c
                inc     al
                add     al,[si].x_lu
                cmp     al,[si].x_rl
                @if     ae
                        ; right edge
                        print_char <0dh>
                        print_char <0ah>
                @else
                        hcursor <1>
                @endif

                ret
endf


;------------------------------
; 文字列を分解
; print_string: print string
; argument:
;               strptr: string pointer
; return:
;               none

func    print_string    <strptr>,<si>
                mov     si,strptr

                @begin
                        lodsb
                        or      al,al
                @while  nz
                        mov     ah,0
                        print_char <ax>
                @repeat
                ret
endf


;--------------------------
; サンプルプログラム
entry:
                @setup
                push    _argc
                push    _argv
                call    _main
exit:
                dos     <4ch,,,>

                .data
wbuf            db      8000 dup (?)

dstr            db      'Hello world',0dh,0ah,0
                .code

func    main    <argc,argv>
                open_window <3,3,20,20,1,<offset wbuf>,ds,0e3h,5>

                @begin
                        mov     ah,6
                        mov     dl,0ffh
                        int     21h
                        or      al,al
                        @if     nz
                                @break
                        @endif
                        print_string <<offset dstr>>
                @again

                close_window <>

                disable_cg <>

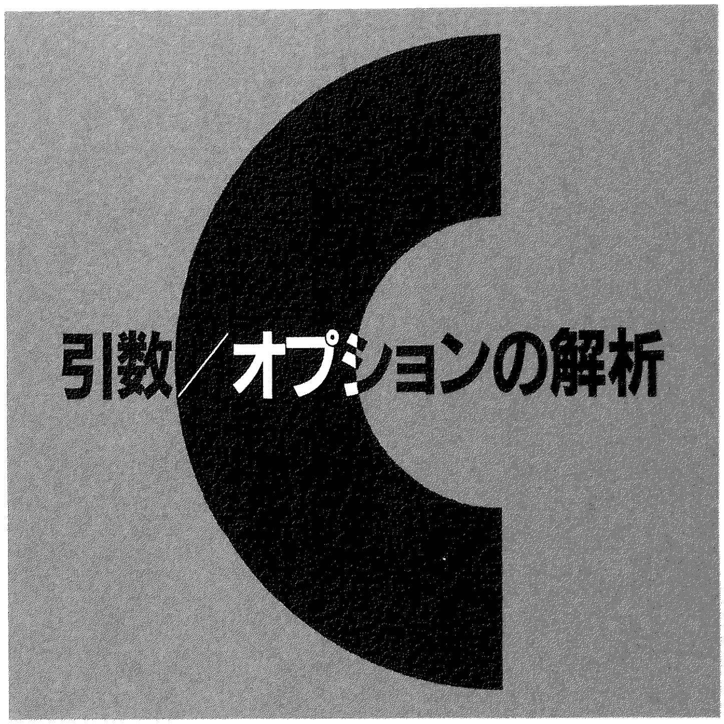
                ret
endf


                end     entry
```

Illustration : Masami Nozue

実践C塾

基礎テクニック篇　第1回



二代目塾頭　大野元久

今月より水居浄氏にかわって大野元久氏を二代目塾頭に迎え，新たに実践C塾を開講する．前回まで6回にわたってMicrosoft C，Turbo C両処理系の特徴，ライブラリの差異について解説を行ってきたが，今回より実用的なアプリケーションを作成するためのC言語の基礎テクニックの解説を開始する．

今回から，このコーナーを担当することになりました．これまでは処理系の比較が中心でしたが，今後は実際的なプログラムについて紹介したいと思います．プログラムの対象は，MS-DOS・PC-9801が中心となります．作成するプログラムも，できるだけ実用性のあるものを目指したいと考えています．もっとも，誌面と時間の都合上，まとまったアプリケーションというよりも，その中で使われるサブルーチン的なものになると思います．

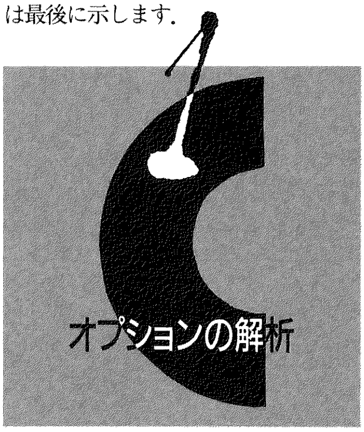
私の常用コンパイラがTurbo C++なので紹介するプログラムもTurbo C++用になりますが，できるかぎりMicrosoft C 6.0でもコンパイルできるように配慮します．これは，両者がMS-DOS用の主流Cコンパイラの最新版であるからです．Turbo C 2.0やMicrosoft C 5.1をお使いの方も多いかもしれませんが，これらについてはとくに配慮はしません．もっとも，多くのプログラムは若干の変更でコンパイルできるでしょう．Zortech C++やLattice Cなどほかのコンパイラについては，執筆時点で最新版がPC-9801に移植されていないことと私自身が使い慣れていないのでいっさい考慮しません．

使用するコンパイラについてくどくどと述べるのは，私の書くプログラムの多くが処理系，つまりTurbo C++に依存したものになるからです．日頃，移植性に配慮しなければならないプログラミングを強いられていないということもありますが，便利な機能や関数群は特定の処理系にしか用意されていないことがほとんどです．私はコンパイラがもっている機能やライブラリ関数で提供されているものは，それが便利なものであれば積極的に利用します．新しい機能というものはとかく問題を起こしがちですが，多くの場合そうした問題以上のメリットが得られます．用意されている機能を使わずに自作したり，まわりくどい解決策を施すというのは虚しいものです．

ちょっと問題を出してみましょう．

「コマンドライン引数のすべてを1文だけで表示してください．」

ここで，1文とはプログラム中にセミコロンが1個しかあらわれないことを意味します．したがって，繰り返し文は使えません．メモリモデルはスモールモデルとします．この問題は，おそらくTurbo C++やMicrosoft Cに依存した問題となるでしょう．たとえば，私が用意した解答はZortech C++では正しく動作しません．解答は最後に示します．



多くのプログラムが，処理方法を切り換えたり，条件を指定するためにコマンドラインオプションを使います．ちょっとしたプログラムを作成する場合は，オプションを解析するためにswitchブロックでcase文を羅列することもありますが，オプションの数を増やすときに不便です．Turbo C++では，getopt.cというオプションを解析するためのサンプルプログラムが提供されていますが，複数の文字列からなるオプションを指定できない，非オプション引数の後ろにオプションを指定できないなどの不満があります．そこで，このコーナーの最初の題材として，オプションの解析ルーチンを作成してみました（リスト1，2，3）．

このモジュールで認識できるコマンドラインオプションの形式は図1のとおりです．

ここで，“progname”は実行ファイルの名前です．

“option”はオプション文字列（1文字以上）です．“optionarg”は，“option”のための引数です．“option”との間にスペースを入れてはいけません．-IA:¥TC¥INCLUDEのような引数をとるオプションの場合に指定します．

オプションを開始するための文字をスイッチ文字と呼び，グローバル変数optdosswとoptswitchの両方を使います．optdosswはDOSのスイッチ文字をあらわし，デフォルトでは自動的にDOSのスイッチ文字を取得するようになっています（通常は‘/’）．optswitchは，DOSのスイッチ文字以外の文字をスイッチ文字として認識したい場合に使うもので，デフォルトでは‘-’になっています．

オプションには，なにも引数を取らないもの，後続する‘+’や‘-’でオプションの意味を切り換えるフラグ形式のもの，文字列を引数としてとるもの，およびこれらを組み合わせたものがあります．引数をとらないオプションは，/ml/ziのように連続して指定することもできます．そのかわり，DOSのスイッチ文字がデフォルトの‘/’のままでは/direc/fileという入力をパス名として受けつけません．DOSのスイッチ文字を認識させたくない場合は，あらかじめグローバル変数optdosswを‘-’のような‘/’以外の文字に設定しておいてください．オプション文字列

は，デフォルトでは大・小文字が区別されますが，グローバル変数optcaseを0にしておけば大・小文字は区別されなくなります．オプション文字列として全角文字列を指定することはできません．

“arg”は，プログラムのための非オプション引数です．非オプション引数とオプション引数はどの順序で指定されてもかまいません．解析ルーチン(getarg)は，指定された順序でこれらを返します．

オプション解析モジュールの使い方

オプション解析モジュールの使い方を説明します．まず，オプション文字列を文字列へのポインタの配列optlistで指定します（リスト4）．optlist[0]は空文字列（""）へのポインタでなければなりません．オプションの個数がn個であれば，optlist[1]～optlist[n]に各オプション文字列へのポインタが設定され，optlist[n+1]にNULLを格納します．

引数は，解析ルーチンgetargを呼び出すたびに1つずつ返されます．引数が非オプション文字列であればgetargは0を返し，文字列へのポインタがoptargに設定されます．引数がオプションであれば，getargは配列optlistへのインデックス値を返します．引数が不正なオプションであれば，インデックスの最大値＋1を返します．このため，インデックス値に対応する列挙型を定義しておくと便利です．列挙型の最後は，不正なオプションをあらわすシンボルを定義しておきます（リスト5）．デフォルトでは，不正なオプションが与えられるとコンソールにエラーメッセージ（"Incorrect command line option"）が表示されます．エラー

リスト1　turboc.h

```
/*
 *  Microsoft C のマクロ、構造体、関数の定義を Turbo C++ 形式に変更する。
 */
#ifndef __TURBOC_H
#define __TURBOC_H

#if !defined(__TURBOC__)

#if !defined(FP_SEG)
#include <dos.h>
#endif

/*
 *  f a r ポインタ用のマクロ
 */
#undef FP_SEG
#undef FP_OFF
#define FP_SEG(fp) ((unsigned)((unsigned long)(fp) >> 16))
#define FP_OFF(fp) ((unsigned)(fp))
#define MK_FP(seg,ofs) (((unsigned long)(unsigned)(seg)<<16) | (unsigned)(ofs))

/*
 *  ワイルドカード検索ルーチン
 */
struct ffblk {
     char ff_reserved[21];
     char ff_attrib;
     unsigned ff_ftime;
     unsigned ff_fdate;
     long ff_fsize;
     char ff_name[13];
};
#define findfirst(path,ffblk,attrib) ¥
     -(int)_dos_findfirst(path,attrib,(struct find_t *)ffblk)
#define findnext(ffblk) -(int)_dos_findnext((struct find_t *)ffblk)

#if !defined(_MAX_PATH)
#include <stdlib.h>
#endif

/*
 *  ファイル名操作ルーチン
 */
#define MAXPATH 80
#define MAXDRIVE 3
#define MAXDIR 66
#define MAXFILE 9
#define MAXEXT 5
#define fnsplit(path,drv,dir,file,ext) _splitpath(path, drv, dir, file, ext)
#define fnmerge(path,drv,dir,file,ext) _makepath(path, drv, dir, file, ext)

#endif /* !defined(__TURBOC__) */

#endif  /* __TURBOC_H */
```

リスト2　option.h

```
/*
 *  option.h: オプションの解析モジュールのためのヘッダファイル
 */

/*
 *  オプション文字列のリスト
 *  （ユーザプログラムで定義）
 */
extern char *optlist[];

/*
 *  オプション解析モジュールで定義されている変数
 */
extern char *optarg;          /* オプションの引数 */
extern int optflag;           /* オプションのフラグ */
extern char optswitch;        /* スイッチ文字 */
extern char optdossw;         /* MS-DOS のスイッチ文字 */
extern int optcase;           /* オプションの大・小文字の区別 */
extern int optwild;           /* ワイルドカードを展開するかどうか */
extern int optenvvar;         /* 環境変数の展開 */
extern int opterr;            /* エラーメッセージを出力するかどうか */
extern int (*optgetc)(void);     /* 1文字取得ルーチンへのポインタ */

/*
 *  各種定数の定義
 */
#define OPTARG_MAX 127        /* オプション文字列や引数の最大長 */

/*
 *  プロトタイプ宣言
 */
void resetarg(void);          /* 解析ルーチンの初期化 */
int getarg(void);             /* 1つの引数/オプションを返す */

int optgetc_arg(void);        /* コマンドライン引数を1文字ずつ返す */
```

リスト3 option.c

```c
/*
 *  option.c: オプションの解析モジュール
 *      written by Motohisa Ohno
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <errno.h>
#include <dos.h>
#if defined(__TURBOC__)
#include <dir.h>
#endif
#include "turboc.h"
#include "option.h"

static unsigned cmdline_pos = 0;    /* ＰＳＰ中の引数の位置 */

/*
 *  コマンドライン引数を１文字ずつ返す
 */
int optgetc_arg(void)
{
    /* コマンドライン引数を最後まで解析したか？ */
    if (cmdline_pos >= *(unsigned char far *)MK_FP(_psp, 0x0080))
        return EOF;
    /* １文字取得 */
    return *(unsigned char far *)MK_FP(_psp, 0x0081 + cmdline_pos++);
}

char *optarg = NULL;            /* 引数／オプションの引数へのポインタ */
int optflag = 1;                /* オプションのフラグ */

char optswitch = '-';           /* スイッチ文字 */
char optdossw = 0;              /* MS-DOS のスイッチ文字 */
int optcase = 1;                /* オプションの大・小文字の区別 */
int optwild = 1;                /* ワイルドカードの展開 */
int optenvvar = 1;              /* 環境変数の展開 */
int opterr = 1;                 /* エラーメッセージの表示 */
int (*optgetc)(void) = optgetc_arg;     /* １文字取得ルーチンへのポインタ */

enum optparse_t {
    PARSE_NORMAL,   /* 新しい引数の解析（通常状態） */
    PARSE_OPTION,   /* オプションの解析中 */
    PARSE_WILD,     /* ワイルドカードの展開中 */
    PARSE_END,      /* 解析終了 */
    PARSE_UNINIT,   /* 未初期化状態 */
};

static enum optparse_t stat = PARSE_UNINIT; /* オプションの解析状態 */
static int optmaxlen = 0;       /* オプション文字列の最大長 */
static int quote = 0;           /* クォート（引用符）の状態 */
static int use_quote;

/*
 *  解析ルーチンを初期化する
 */
void resetarg(void)
{
    union REGS regs;
    int ndx, n;

    /* 解析状態を通常状態にリセット */
    stat = PARSE_NORMAL;

    /* DOS のスイッチ文字を取得する */
    if (optdossw == 0) {
        regs.x.ax = 0x3700;
        intdos(&regs, &regs);
        optdossw = regs.h.dl;
        if (optdossw == optswitch) optswitch = 0;
    }

    /* オプション文字列の最大長 */
    if (optmaxlen == 0) {
        for (ndx = 1; optlist[ndx]; ndx++) {
            /* 指定された文字列の長さ － 形式をあらわす文字の長さ */
            n = strlen(optlist[ndx]) - strspn(optlist[ndx], "+:");
            if (optmaxlen < n) optmaxlen = n;
        }
    }

    /* クォートの状態をリセット */
    quote = 0;
}

static int unget_char = 0;

/*
 *  １文字取得ルーチン（環境変数の展開を行なう）
 */
static int optgetce(void)
{
    static unsigned char envstr[OPTARG_MAX+2];
    static int envstrp = 0;
    static int enveof = 0;
    char *ep;
    int c;

    /* １文字プッシュバックに対応 */
    if (unget_char) {
        c = unget_char;
        unget_char = 0;
        return c;
    }

    /* 展開した環境変数を１文字ずつ返す */
    if (envstrp) {
        if (envstr[envstrp]) return envstr[envstrp++];
        envstrp = 0;
    }
    /* すでに EOF を取得していたら EOF を返す */
    if (enveof) {
        enveof = 0;
        return EOF;
    }
    c = optgetc();
    if (optenvvar == 0 || c != '%') return c;
    envstr[0] = '%';
    for (envstrp = 1;
            (c = optgetc()) != '%' && isalnum((unsigned char)c)
                                && envstrp < OPTARG_MAX; envstrp++) {
            envstr[envstrp] = c;
    }
    envstr[envstrp] = 0;
    if (c == '%') {
        if (envstrp == 1) return '%';
        ep = getenv(envstr + 1);
        strncpy(envstr, ep ? ep : "", OPTARG_MAX);
        envstr[OPTARG_MAX] = 0;
    } else if (c == EOF) {
        enveof = 1;
    } else {
        envstr[envstrp++] = c;
        envstr[envstrp] = 0;
    }
    envstrp = 0;
    return envstr[envstrp++];
}

/*
 *  １文字プッシュバックルーチン（optgetce対応）
 */
static void optungetce(int c)
{
    unget_char = c;
}

/*
 *  １文字取得ルーチンの拡張
 *      戻り値：    ＥＯＦ＝これ以上文字はない
 *                  ０＝空白文字（引数の区切り）
 *                  それ以外＝通常の文字（引用符で囲まれた空白文字を含む）
 */
static int arggetc(void)
{
    int c;

    do {
        c = optgetce();
        if (c == EOF) {
            stat = PARSE_END;
            return c;
        }
        if (isspace(c) && quote == 0) {
            while (isspace(c)) {
                c = optgetce();
                if (c == EOF) {
                    stat = PARSE_END;
                    return c;
                }
            }
            stat = PARSE_NORMAL;
            optungetce(c);
            return 0;
        }
        if (c == '¥¥') {
            c = optgetce();
            if (c == EOF) {
                stat = PARSE_END;
                return '¥¥';
            }
            if (c != '"' && c != '¥¥') {
                optungetce(c);
                c = '¥¥';
            }
            return c;
        }
        if (c == '"') {
            quote = !quote;
            use_quote = 1;
        }
    } while (c == '"');
    return c;
}

/*
 *  １文字プッシュバックルーチンの拡張
 */
static void argungetc(int c)
{
    optungetce(c);
}

/*
 *  エラーメッセージを表示するためのルーチン
 */
static void message(char *msg)
{
    while (*msg) {
        bdos(2, *msg, 0);
        ++msg;
    }
}

/*
```

```c
 *  解析の実行。1つの引数／オプションを返す。
 */
int getarg(void)
{
    static char buf[OPTARG_MAX+1];  /* 引数文字列の格納領域 */
    static struct ffblk find;       /* ワイルドカード解析情報 */
    static char wildpath[MAXPATH];  /* ワイルドカードパス */
    int c, i, n, ndx;

    if (stat == PARSE_UNINIT) resetarg();

    /* 解析が終了していれば EOF を返す */
    if (stat == PARSE_END) {
        optarg = NULL;
        optflag = 1;
        return EOF;
    }

    /* ワイルドカード検索の続き */
    if (stat == PARSE_WILD) {
        /* ワイルドカード検索の実行 */
        if (findnext(&find) == 0) {     /* 検索成功 */
            strcpy(buf, wildpath);
            strcat(buf, find.ff_name);
            optarg = buf;
            optflag = 1;
            return 0;
        }
        /* 検索に失敗したら、次の引数を解析する */
        stat = PARSE_NORMAL;
    }

    /* 通常の引数解析 */
normal:;
    if (stat == PARSE_NORMAL) {
        /* 1文字取得 */
        do {
            c = arggetc();
            /* 文字がなくなったら EOF を返す */
            if (c == EOF) {
                optarg = NULL;
                optflag = 1;
                return c;
            }
            /* 空白文字はスキップする */
        } while (c == 0);

        /* 読み込んだ文字がスイッチ文字であれば、オプション解析 */
        /* そうでなければ、引数文字列の先頭へ格納 */
        if (c == optdossw || optswitch && c == optswitch)
            stat = PARSE_OPTION;
        else {
            use_quote = 0;

            i = 0;
            do {
                if (i < OPTARG_MAX) buf[i++] = c;
                c = arggetc();
            } while (c > 0);
            buf[i] = 0;
            if (optwild && use_quote == 0
                    && (strchr(buf, '*') || strchr(buf, '?'))) {
                char drv[MAXDRIVE], dir[MAXDIR];

                fnsplit(buf, drv, dir, NULL, NULL);
                strcpy(wildpath, drv);
                strcat(wildpath, dir);
                /* ワイルドカード検索 */
                if (findfirst(buf, &find, 0) == 0) {    /* 検索成功 */
                    stat = PARSE_WILD;
                    strcpy(buf, wildpath);
                    strcat(buf, find.ff_name);
                    optarg = buf;
                    optflag = 1;
                    return 0;
                }
                /* 検索が失敗したら通常の文字列として扱う */
            }
            optarg = buf;
            optflag = 1;
            return 0;
        }
    }

    /* エラーメッセージ用に、buf の先頭にスイッチ文字を格納しておく */
    buf[0] = optswitch ? optswitch : optdossw;
    for (n = 1; n <= optmaxlen; n++) {
        /* 引数を1文字ずつ取得して、オプション文字列に */
        /* 一致するかどうか解析する */
        c = arggetc();
        if (c == EOF || c == 0) break;
        if (n == 1 && (c == optdossw || optswitch && c == optswitch)) {
            c = arggetc();
            if (c == EOF || c == 0) break;
        }
        buf[n] = c;
        buf[n+1] = 0;
        for (ndx = 1; optlist[ndx]; ndx++) {
            /* オプションの形式をあらわす文字の長さ */
            int len = strspn(optlist[ndx], "+:");

            /* コマンドライン引数とオプション文字列とを比較 */
            if (((optcase ? strcmp : stricmp)(buf+1, optlist[ndx]+len)) == 0) {
                /* 一致したらそのオプションを解析 */
                char *p;
                int flagtype = 0;
                int argtype = 0;

                /* オプションの形式を取得 */
                for (p = optlist[ndx]; *p == '+' || *p == ':'; p++) {
                    if (*p == '+')
                        flagtype = 1;
                    else
                        argtype = 1;
                }
                if (flagtype) {
                    /* フラグ形式であれば直後の1文字を読み込む */
                    c = arggetc();
                    if (c == '+' || c == '-') {
                        /* '+' か '-' であれば、optflag を設定 */
                        optflag = (c == '+');
                    } else {
                        /* そうでなければ、文字を返して optflag を1に設定 */
                        argungetc(c);
                        optflag = 1;
                    }
                } else {
                    /* フラグ形式でなければ、念のため optflag を1に設定 */
                    optflag = 1;
                }
                if (argtype) {
                    /* 引数をとる形式であれば、後ろの文字列を読み込む */
                    i = 0;
                    while ((c = arggetc()) != EOF) {
                        if (c == 0) break;
                        if (i < OPTARG_MAX) buf[i++] = c;
                    }
                    buf[i] = 0;
                    optarg = buf;
                } else {
                    /* 引数をとる形式でなければ、optarg を NULL に設定 */
                    optarg = NULL;
                }
                return ndx;
            }
        }
    }

    if (n <= 1) {
        /* 引数がなくなっていたら、通常の解析状態へ移行 */
        stat = PARSE_NORMAL;
        goto normal;
    }

    /* どのオプション文字列にも該当しなかった引数をエラーとする */
    if (n > optmaxlen) {
        while ((c = arggetc()) != EOF) {
            if (c == 0) break;
            if (n < OPTARG_MAX) buf[n++] = c;
        }
        buf[n] = 0;
    }

    optarg = buf;
    errno = EINVAL;
    if (opterr) {
        /* perror("get command line option"); */
        message("Incorrect command line option: ");
        message(buf);
        message("YrYn");
    }
    return ndx; /* ndx は、オプションリストのインデックスの最大値+1 */
}
```

図1 コマンドラインオプションの形式

```
progname [-option[optionarg]]... [arg ...]
```

リスト4 配列optlistの定義例

```c
char *optlist[] = {
    "",
    ":A",    /* オプションA（引数あり） */
    "B",     /* オプションB（引数なし） */
    "help",  /* ヘルプ */
    NULL,
};
```

メッセージの表示を抑止するためには，グローバル変数opterrを0にしておきます．

デフォルトでは，非オプション文字列がDOSのワイルドカード(*，?)を含む場合はワイルドカードが展開され，一致するファイル名が1つずつ返されます．ワイルドカードの展開を抑止するためには，グローバル変数optwildを0にしておきます．

通常，コマンドラインは空白文字で区切られますが，引用符と引用符で囲まれた文字列は空白文字を含むことができます．また，引用符でくくられた文字列はワイルドカードが展開されません．

円記号(¥)は円記号自身と引用符のためのエスケープ文字として機能します．円記号の後ろにこれらの文字がくるときは，これらの文字は通常の文字として扱われます．円記号の後ろがこれらの文字でなければ，円記号はそのまま文字列に残ります．

'%'で囲まれた文字列は環境変数とみなされ，展開してから渡されます．環境変数は英数字のみが許され，全角文字や制御文字は使えません．環境変数は，引用符に囲まれた文字列の中でも展開されます．環境変数を展開しない場合は，グローバル変数optenvvarを0にしておきます．

引数がこれ以上存在しない場合は，getargはEOFを返します．改めてオプションを解析する場合は，resetargを呼び出して解析状態をリセットしなければなりません．一度解析が終了してしまうと，resetargが呼び出されないかぎりgetargはEOFを返し続けます．

オプション文字列の先頭が'+'であれば，そのオプションはフラグ形式であることを示します．フラグ形式のオプションでは，オプション文字列の後ろに何も指定されないか'+'が指定された場合にはグローバル変数optflagに1を，'-'が指定された場合には0を設定します．

オプション文字列の先頭が':'であれば，そのオプションが引数をとるオプションであることを示します．オプションのための引数は，そのオプションと空白を空けずに指定します．この引数文字列へのポインタは，グローバル変数optargに返されます．'+'と':'は組み合わせて使用することができま

リスト5　列挙型の定義例

```
enum myoption_t {
     OPT_ARG,      /* 非オプション文字列 */
     OPT_A,        /* オプションA */
     OPT_B,        /* オプションB */
     OPT_HELP,     /* ヘルプ */
     OPT_ILL,      /* 不正なオプション */
};
```

リスト6　オプション解析モジュールを使用したプログラム例 (opttest.c)

```
#include <stdio.h>
#include "option.h"

char *optlist[] = {
     "",
     ":A",    /* オプションA (引数あり) */
     "B",     /* オプションB (引数なし) */
     "+f",    /* オプションf (フラグ形式) */
     ":+w",   /* オプションw (引数あり、フラグ形式) */
     "h",     /* ヘルプ */
     "?",     /* ヘルプ */
     "help",  /* ヘルプ */
     NULL,
};

enum myoption_t {
     OPT_ARG,      /* 非オプション文字列 */
     OPT_A,        /* オプションA */
     OPT_B,        /* オプションB */
     OPT_f,        /* オプションf */
     OPT_w,        /* オプションw */
     OPT_HELP,     /* ヘルプ */
     OPT_HELP2,    /* ヘルプ */
     OPT_HELP3,    /* ヘルプ */
     OPT_ILL,      /* 不正なオプション */
};

void parseopt()
{
     int s;

     while ((s = getarg()) != EOF) {
          switch (s) {
          case OPT_ARG:
               printf("argument: %s¥n", optarg);
               break;
          case OPT_A:
               printf("option A(%s)¥n", optarg);
               break;
          case OPT_B:
               printf("option B¥n");
               break;
          case OPT_f:
               printf("option f(%d)¥n", optflag);
               break;
          case OPT_w:
               printf("option w(%d,%s)¥n", optflag, optarg);
               break;
          case OPT_HELP:
          case OPT_HELP2:
          case OPT_HELP3:
               printf("help option¥n");
               break;
          case OPT_ILL:
               printf("Illegal option(%s)¥n", optarg);
               break;
          }
     }
}

void main()
{
     parseopt();
}
```

す．

リスト6に，このモジュールを使用したプログラム例（opttest. c）を示します．このプログラムに図2のコマンドライン引数を指定すると，図3のように表示されます．

図2 コマンドライン引数

```
-Aarg -f-B first -w-foo -wbar second
```

図3 opttest.cの結果

```
option A(arg)
option f(0)
option B
argument: first
option w(0,foo)
option w(1,bar)
argument: second
```

プログラムの解説

簡単にプログラムの説明をします．

turboc.h

turboc. hは，Microsoft CでTurbo C++がサポートしているマクロや関数を使用するためのヘッダファイルです．

Microsoft C形式のfarポインタを扱うマクロは，farポインタでないポインタやアドレスを渡せないという制限があるため，Turbo C++形式に変更しています．こうすると，farポインタのセグメント部やオフセット部だけを変更することができなくなりますが，このような処理は浮動小数値の仮数部や指数部だけを変更するような奇異なイメージがあるためもともと私は使用しません．ワイルドカード検索とファイル名操作のための関数は，それぞれほぼ等価なMicrosoft Cの関数を呼び出すようにマクロ定義しています．

turboc. hが置き換える内容は，option. cのためには十分ですが，必ずしも完全に仕様の置き換えを行っているわけではないことに注意してください．たとえば，Turbo C++のfindfirstは検索に失敗すると－1を返しますが，Microsoft Cの_dos_findfirstはDOSのエラーコードを返します．このため，マクロ定義されたfindfirstの戻り値と－1を比較して検索の失敗を判断してはいけません．

このほかにも，option. cのoptgetc_arg関数で使われている_psp変数は，Microsoft Cではstdlib. hでしか定義されていませんが，Turbo C++ではdos. hやprocess. hでも定義されているなど，両者の間にはさまざまな違いがあります（両者の違いの詳細については，本誌2～6月号の本連載をご覧ください）．turboc. hは今後も使用する予定ですが，必要に応じて改良することになるでしょう．また，turboc. hで吸収できない違いについては，定義済みマクロ__TURBOC__が定義されているかどうかを判断してソースコードを変更します．

option.h

option. hは，このモジュールを利用するためのヘッダファイルです．ここには，解析の方法を制御するための前述の変数や関数のプロトタイプ宣言が含まれています．

option.c

option. cは，実際にオプションを解析するためのモジュールです．

optgetc_argは，_psp変数で与えられるセグメントにあるプログラムセグメントプレフィクス（PSP）にあるコマンドライン引数を1文字ずつ返す関数です．この解析モジュールでは，mainに与えられるargcやargvをいっさい使用していません．解析ルーチンは，optgetcという1文字取得へのポインタを介して引数を1文字ずつ間接的に処理しています．デフォルトでは，この変数にoptgetc_argが設定されておりコマンドライン引数の処理を行いますが，このポインタにほかの関数へのポインタを設定することでコマンドライン以外のオプションや引数を解析できるようになります．リスト7は，環境変数OPTIONに設定されたオプションを解析したあとでコマンドライン引数を解析するようにopttest. cを書き換えたプログラム例（optenv. c）です．

optarg，optflagは，引数やオプションの内容を返すためのグローバル変数です．optswitch～optgetcは，オプションを解析するための設定をあらわします．

resetargは，オプションの解析状態を初期化する関数です．解析状態を通常状態に設定したり，DOSのスイッチ文字やオプション文字列の最大長を求めます．

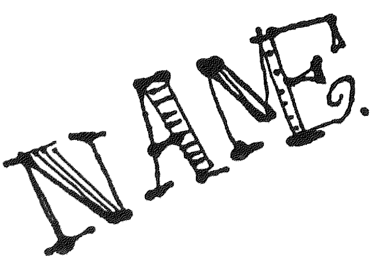
optgetce/optungetceは，1文字取得ルーチンへのポインタoptgetcを呼び出して環境変数を展開しながら1文字ずつ返す関数です．‘%’で囲まれた環境変数の展開は，この段階で行われるため引用符や円記号によるエスケープはすべて環境変数の展開後に処理されます．arggetcは，optgetce/optungetceを呼び出して引用符や円記号によるエスケープシーケンスを処理しながら1文字ずつ返す関数です．このように文字の処理を段階的に行うこと

で，文字列に対する処理を機能ごとにわけています．

messageは，エラーメッセージを表示するための簡易ルーチンです．本来，エラーメッセージは標準エラー出力に出力するべきかもしれませんが，小さなプログラムでの実行サイズの節約のためにこのような簡易ルーチンを使用しています．もっとも，この解析モジュール自体は数KBのサイズを必要とするためあまり小さなアプリケーションには向いていないかもしれません．

getargは，解析モジュールのおもな処理を行う関数です．解析状態を，未初期化状態，通常状態，オプションの解析中，ワイルドカードの展開中，解析終了の５段階に分割して処理しています．未初期化状態でgetargが呼び出された場合は，初期設定を行うためにresetargを呼び出します．解析が終了したあとで，ほかの解析を行う場合は，再度resetargを呼び出さなければなりません．

実のところ，この解析ルーチンの効率はあまりよくないのですが，オプションの解析などプログラムのはじめに１回だけ実行するものですし，よほど長いオプション文字列を使うのでなければ問題はないでしょう．モジュールの大きさについても，ワイルドカードや環境変数の展開など，いろいろな機能を取り込んでしまっているために若干大きくなっています．



リスト7　optenv.c

```
#include <stdio.h>
#include <stdlib.h>
#include "option.h"

char *optlist[] = {
    "",
    ":A",   /* オプションA（引数あり） */
    "B",    /* オプションB（引数なし） */
    "+f",   /* オプションf（フラグ形式） */
    ":+w",  /* オプションw（引数あり、フラグ形式） */
    "h",    /* ヘルプ */
    "?",    /* ヘルプ */
    "help", /* ヘルプ */
    NULL,
};

enum myoption_t {
    OPT_ARG,    /* 非オプション文字列 */
    OPT_A,      /* オプションA */
    OPT_B,      /* オプションB */
    OPT_f,      /* オプションf */
    OPT_w,      /* オプションw */
    OPT_HELP,   /* ヘルプ */
    OPT_HELP2,  /* ヘルプ */
    OPT_HELP3,  /* ヘルプ */
    OPT_ILL,    /* 不正なオプション */
};

void parseopt()
{
    int s;

    resetarg();
    while ((s = getarg()) != EOF) {
        switch (s) {
        case OPT_ARG:
            printf("argument: %s¥n", optarg);
            break;
        case OPT_A:
            printf("option A(%s)¥n", optarg);
            break;
        case OPT_B:
            printf("option B¥n");
            break;
        case OPT_f:
            printf("option f(%d)¥n", optflag);
            break;
        case OPT_w:
            printf("option w(%d,%s)¥n", optflag, optarg);
            break;
        case OPT_HELP:
        case OPT_HELP2:
        case OPT_HELP3:
            printf("help option¥n");
            break;
        case OPT_ILL:
            printf("Illegal option(%s)¥n", optarg);
            break;
        }
    }
}

static char *optenv_str;
static int optenv_ndx = 0;

int optgetc_env(void)
{
    if (optenv_str[optenv_ndx] == 0) return EOF;
    return (unsigned char)optenv_str[optenv_ndx++];
}

void main()
{
    optenv_str = getenv("OPTION");
    if (optenv_str) {
        optgetc = optgetc_env;
        parseopt();
    }
    optgetc = optgetc_arg;
    parseopt();
}
```



起動パス／ファイル名の取得

ときとして，プログラムがどのドライブやディレクトリから呼び出されたかを調べておかなければならないことがあります．ある種のプログラムでは，自分が利用するためのデータを自分が呼び出されたドライブ／ディレクトリから参照するようになっています．

たとえば，Turbo C++のコマンドラインコンパイラはコマンドライン引数を解析する前にあらかじめ用意されたコンフィギュレーションファイルを起動パスから参照します．

MS-DOSのバージョン3.0以降では，簡単に起動パスを調べることができます．関数mainに与えられる第2引数argvの先頭には，起動パスを含む完全なファイル名へのポインタが与えられます．この引数argvと等価なグローバル変数として，Turbo C++では_argvを，Microsoft Cでは_argv（非公開変数）を使用できますので，起動パス／ファイル名はリスト8のような簡単なプログラムで設定できます．

しかし，廃れたとはいえDOSのバージョン2.xを使用している人もいます．せっかくですから，このバージョンにも対応してみましょう．バージョン2.xでは，argv[0]には空文字列が与えられるだけで，自分自身のファイル名を取得することができないため，あらかじめファイル名を設定しておくことにします．そして，カレントドラ

リスト8　startp.c

```
#include <stdlib.h>
#if defined(__TURBOC__)
#include <dir.h>
#endif
#include "turboc.h"

char *startpath = "";
char *progname = "";

/*
 *  起動パスとプログラム名（拡張子を除く）を設定する
 */
void setstartpath(void)
{
        static char path[MAXPATH];
        static char myname[MAXFILE];
        char dir[MAXDIR], drv[MAXDRIVE];

#if defined(__TURBOC__)
        /* Turbo C++ では、_argv と main に渡される argv とは同じ */
        fnsplit(_argv[0], drv, dir, myname, NULL);
#else
        /* Microsoft C では、非公開変数 __argv を使用する */
        fnsplit(__argv[0], drv, dir, myname, NULL);
#endif
        fnmerge(path, drv, dir, NULL, NULL);
        startpath = path;
        progname = myname;
}

```

リスト9　startp.c（改良版）

```
/*
 *  startp.c: 起動パス／プログラム名設定モジュール
 *      written by Motohisa Ohno
 */
#include <stdlib.h>
#include <io.h>
#include <dos.h>
#if defined(__TURBOC__)
#include <dir.h>
#else
#include <direct.h>
#endif
#include "turboc.h"

char *startpath = "";
char *progname = "";

/*
 *  起動パスとプログラム名（拡張子を除く）を設定する.
 *  exename には、プログラムの実行ファイル名を与える。
 */
void setstartpath(char *exename)
{
        static char path[MAXPATH];
        static char myname[MAXFILE];
        char drv[MAXDRIVE], dir[MAXDIR];
        char *p;
        int n;
#if !defined(__TURBOC__)
        extern char **__argv;
#endif

        if ((unsigned char)bdos(0x30, 0, 0) < 3) {
            /* MS-DOS バージョン３．０より前のバージョン */
            /* あらかじめ exename に自分自身のファイル名を */
            /* 指定しておかなければならりません。 */
            if (exename[0] == 0) {
                /* exename が指定されていなければ */
                /* そのまま戻る */
                return;
            }
            fnsplit(exename, NULL, NULL, myname, NULL);
            progname = myname;
#if defined(__TURBOC__)
            getcwd(path, MAXPATH);
            n = strlen(path);
            if (n && path[n-1] != '¥¥') strcat(path, "¥¥");
            if (access(exename, 0) == 0) {
                /* カレントディレクトリにファイルが存在すれば */
                /* 起動パスを設定して戻る */
                startpath = path;
                return;
            }
            /* 環境変数 PATH を使ってファイルを検索 */
            p = searchpath(exename);
            if (p)
                strcpy(path, p);
            else
                path[0] = 0;
#else
            /* Microsoft C の場合 */
            /* 環境変数 PATH を使ってファイルを検索 */
            _searchenv(exename, "PATH", path);
#endif
            if (path[0] == 0) return;
            /* 検索が成功したらドライブ／ディレクトリだけを取り出して戻る */
            fnsplit(path, drv, dir, NULL, NULL);
            fnmerge(path, drv, dir, NULL, NULL);
            startpath = path;
            return;
        }
        /* MS-DOS バージョン３．０以降 */
#if defined(__TURBOC__)
        /* Turbo C++ では、_argv と main に渡される argv とは同じ */
        fnsplit(_argv[0], drv, dir, myname, NULL);
#else
        /* Microsoft C では、非公開変数 __argv を使用する */
        fnsplit(__argv[0], drv, dir, myname, NULL);
#endif
        fnmerge(path, drv, dir, NULL, NULL);
        startpath = path;
        if (exename[0]) fnsplit(exename, NULL, NULL, myname, NULL);
        progname = myname;
}

```

イブ／ディレクトリに自分のファイル名が存在するかどうか確かめ，存在しなければ環境変数PATHを使って自分のファイル名を検索します．この検索には，Turbo C++ならsearchpath，Microsoft Cなら_searchenvを使うことができます．

バージョン2.xのDOSでは，コマンドラインから直接起動パスを指定してコマンドを実行することができないので，ファイル名が変更されないかぎりこの方法で十分実用的です．ドライブ名だけを指定された場合は完全には対応できませんがいたしかたないでしょう．

バージョン2.xのDOSに対応したstartpath関数とこの関数を使用したプログラム例をリスト9，10に示します．改良されたstartpath関数は，引数として設定しておくプログラム名を与えます．こうすることで，実際のファイル名が変更されたかどうかに関わらず設定したファイル名がそのプログラム名として使用されます．引数に空文字列（""）が与えられると，DOSバージョン3.x以降にかぎり起動ファイル名がプログラム名として使用されます．

リスト10　startpを使用したプログラム例（stest.c）

```
#include <stdio.h>
#include <io.h>
#if defined(__TURBOC__)
#include <dir.h>
#endif
#include "turboc.h"

extern char *startpath, *progname;

void main()
{
    char datapath[MAXPATH];

    /* プログラム名を設定 */
    setstartpath("STEST.EXE");
    /* 起動パスとプログラム名の表示 */
    printf("startpath = %s¥n", startpath);
    printf("progname = %s¥n", progname);

    /* プログラムのためのデータファイルの位置 */
    fnmerge(datapath, NULL, NULL, progname, ".DAT");
    if (access(datapath, 0) < 0) {
        /* カレントディレクトリにファイルがなければ起動パスを参照 */
        strcpy(datapath, startpath);
        strcat(datapath, progname);
        strcat(datapath, ".DAT");
    }
    printf("datapath = %s¥n", datapath);
}

```

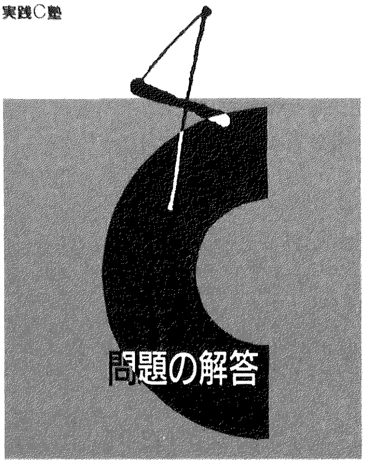


「オプションの解析」「起動パス／ファイル名の取得」で作成したルーチンを使って，プログラムのためのデフォルトのコンフィギュレーションファイルを読み込むルーチンを作成してみます(リスト11)．オプションの解析ルーチンは，リスト6（opttest.c）と同じなので省略しています．

まず，setstartpathでプログラムの起動パスとファイル名を指定します．オプションを解析するreadconfigルーチンでは，カレントディレクトリにコンフィギュレーションファイル（“ファイル名.CFG”）が存在するかどうかを確認し，なければ起動パスに存在するかどうか確認します．コンフィギュレーションファイルが存在すれば，それをオープンして，そこから1文字取得する関数へのポインタを変数optgetcに代入し，オプションの解析を行います．そのあと，コマンドライン引数を解析するためのoptgetc_arg/optungetc_argをこれらの変数に代入して解析を行います．

コンフィギュレーションファイルを利用したオプション解析は，Turboシリーズのコンパイラが採用している手法です．Microsoft系のコンパイラでは，デフォルトのオプションは環境変数から参照することになっていますが，ヘッダファイルやライブラリのあるディレクトリ情報を環境変数を使って設定すると，環境変数領域の消費量が大きくなってしまったり，環境変数名がほかのアプリケーションと競合してしまうことも考えられます．逆にTurboシリーズのようなコンフィギュレーションファイルによる方法では，バッチファイルで簡単に環境を切り換えたりすることができなくなります．

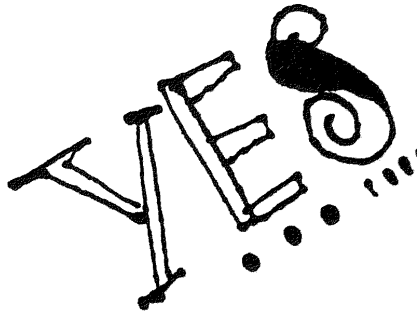
いずれも一長一短ですが，今回紹介したオプション解析モジュールは環境変数の展開をサポートしていますので，コンフィギュレーションファイルを使ったほうが若干柔軟性が高いかもしれません．もっとも，作成するアプリケーションの大きさや目的によって，どちらが望ましいかは変わってきますから，この判断は皆さんにお任せします．環境変数を使用する場合は，「オプションの解析」で示したプログラム例（リスト7）を参考にしてください．もちろん両方を使うこともできるでしょうが，かえって混乱を招くかもしれません．



まず，最初の問題の解答を示しておきます(リスト12, 13)．いずれも，PSPセグメントのオフセット0x80にある文字数だけ，オフセット0x81以降にある文字列を表示するというものです．

printfはよく使われる関数ですが，*指定はあまりなじみがないかもしれません．これは，引数の値を桁幅として使うためのANSIで規定されている書式です．Zortech C++で動作しないというのは，“%Fs”という書式がサポートされていないためです．これは，ポインタを引数とする書式指定の対象がfarポインタであることを示すもので，もちろんANSIの規格ではなくZortech C++が悪いということではありません．

連載の初回ということもあり，うまく解説できているかどうか自信がないのですが，今後もこうした実際的なプログラムの紹介を続ける予定です．バグの報告やご意見などをお待ちしております．



リスト11　readcfg.c

```
/*
 *  readcfg.c: コンフィギュレーションファイルを読み込むプログラム例
 *      written by Motohisa Ohno
 */
#include <stdio.h>
#if defined(__TURBOC__)
#include <dir.h>
#endif
#include "option.h"

char *optlist[] = {
    "",
    ":A",    /* オプションA（引数あり） */
    "B",     /* オプションB（引数なし） */
    "+f",    /* オプションf（フラグ形式） */
    ":+w",   /* オプションw（引数あり、フラグ形式） */
    "h",     /* ヘルプ */
    "?",     /* ヘルプ */
    "help",  /* ヘルプ */
    NULL,
};

enum myoption_t {
    OPT_ARG,     /* 非オプション文字列 */
    OPT_A,       /* オプションA */
    OPT_B,       /* オプションB */
    OPT_f,       /* オプションf */
    OPT_w,       /* オプションw */
    OPT_HELP,    /* ヘルプ */
    OPT_HELP2,   /* ヘルプ */
    OPT_HELP3,   /* ヘルプ */
    OPT_ILL,     /* 不正なオプション */
};

void parseopt()
{
    int s;

    resetarg();
    while ((s = getarg()) != EOF) {
        switch (s) {
        case OPT_ARG:
            printf("argument: %s¥n", optarg);
            break;
        case OPT_A:
            printf("option A(%s)¥n", optarg);
            break;
        case OPT_B:
            printf("option B¥n");
            break;
        case OPT_f:
            printf("option f(%d)¥n", optflag);
            break;
        case OPT_w:
            printf("option w(%d,%s)¥n", optflag, optarg);
            break;
        case OPT_HELP:
        case OPT_HELP2:
        case OPT_HELP3:
            printf("help option¥n");
            break;
        case OPT_ILL:
            printf("Illegal option(%s)¥n", optarg);
            break;
        }
    }
}

extern char *startpath, *progname;

FILE *fpcfg;

int optgetc_cfg(void)
{
    return fgetc(fpcfg);
}

void readconfig(void)
{
    char cfgpath[MAXPATH];

    fnmerge(cfgpath, NULL, NULL, progname, ".CFG");
    if (access(cfgpath, 0) != 0) {
        strcpy(cfgpath, startpath);
        strcat(cfgpath, progname);
        strcat(cfgpath, ".CFG");
    }
    fpcfg = fopen(cfgpath, "r");
    if (fpcfg != NULL) {
        /* コンフィギュレーションファイルがオープンできれば */
        /* その内容を解析する */
        optgetc = optgetc_cfg;
        parseopt();
        fclose(fpcfg);
    }
    /* コマンドラインで指定された内容を解析する */
    optgetc = optgetc_arg;
    parseopt();
}

void main()
{
    setstartpath("READCFG.EXE");
    readconfig();
}
```

リスト12　Turbo C++の場合

```
#include <stdio.h>
#include <dos.h>

void main()
{
    printf("%.*Fs¥n", *(char far *)MK_FP(_psp, 0x80), MK_FP(_psp, 0x81));
}
```

リスト13　Microsoft Cの場合

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    printf("%.*Fs¥n", *(char far *)(((unsigned long)_psp << 16) + 0x80),
        (char far *)(((unsigned long)_psp << 16) + 0x81));
}
```

情報処理技術者試験
第2種合格！トレーニング
第4回　ソフトウェアの基礎II

日本工学院専門学校　長澤洋一郎

ファイル編成

索引順次編成ファイル

任意のレコードがアクセスできるように，順次編成ファイルに，索引を持たせたファイル編成です．

順次アクセスと直接アクセスの両方が行え，マスタファイルとしてよく用いられます．

■ファイルの構造

このファイルは，構造に大きな特徴があります．

データが記録される領域のほかに，データをアクセスするための索引域と，追加によってファイルからあふれたレコードを記録するあふれ域の3つで構成されています．

① 基本データ域（プライム域）

順次編成ファイルに相当し，ファイルを構成するレコードが記録されています．

このファイルのレコードは必ずキー項目を持っていて，レコードはこのキー項目の昇順に記録されています．

② 索引域（インデックス域）

レコードが持っているキー項目と，そのレコードが記録されているアドレス（ポインタ）が，対応づけられて記録されています．

本の巻頭にある目次のような役割をします．

磁気ディスクの構造にあわせて，マスタ索引，シリンダ索引，トラック索引の3つに細分されています．

③ あふれ域（オーバーフロー域）

レコードを追加したとき，基本データ域に入りきらなかったレコードが記録されます．

シリンダごとに作成され，各シリンダからあふれたレコードを記録する「シリンダあふれ域」と，シリンダあふれ域からあふれたレコードを記録する「独立あふれ域」（ゼネラルオーバーフロー）の2つがあります．

■レコードへのアクセス

ファイルの先頭や途中のレコードから，レコードが記録されている順番にアクセスする順次アクセスと，キー項目を指定して，任意のレコードだけをアクセスする直接アクセスの2つが可能です．

■その他の特徴

磁気ディスクなどの，直接アクセスが行える媒体だけにファイルを作成できます．

直接編成ファイル

より高速な直接アクセスができるように考えられたファイル編成です．

おもに，オンラインシステムなど，高速レスポンスが求められる場合に用いられます．アクセスの方法に特徴があります．

■アドレス方式

レコードに持たせたキー項目の値や，レコード内の特定の項目の値をもとにしてアドレスを求め，レコードをアクセスします．

アドレスの求め方には，直接アドレス方式と間接アドレス方式の2つがあります．

① 直接アドレス方式

特定の項目（従業員番号，伝票番号など）の値をそのままアドレスとする方式です．

この方式で作成したファイルをとくに，相対編成と呼ぶ場合があります．

② 間接アドレス方式

レコードにキーを持たせ，このキーの値を変換（ハッシング，ランダマイジング）した答えをアドレスとする方式です．

代表的な変換法として「重ね合わせ法」，「基数変換法」，「除算法」などがあります．

■レコードへのアクセス

① 直接アドレス方式の場合

レコードがファイルの先頭から順番に記録されているため，直接アクセスだけでなく順次アクセスも可能です．

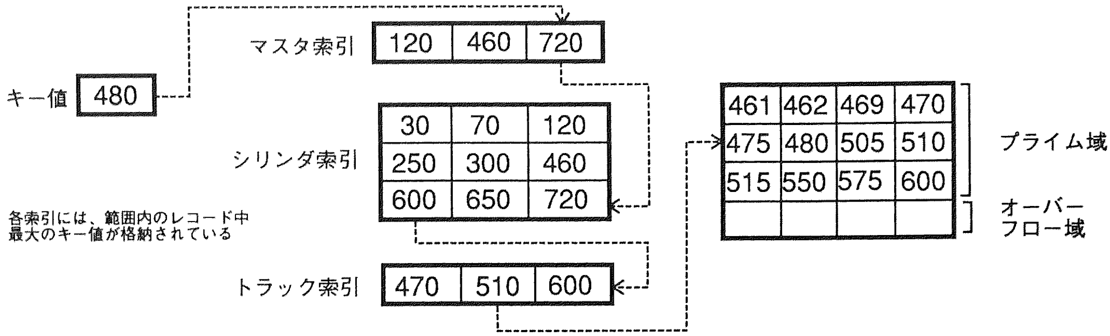
② 間接アドレス方式の場合

キー項目の変換によって特定レコードをアクセスする直接アクセスが可能です．

■その他の特徴

① アドレスの取得は演算で行うので，索引域を検索する索引順次編成より，直接アクセスのアクセス効率がいいです．

② 間接アドレス方式を用いた場合，複数のキーが同じアドレスに変換されることがありえます．同じアドレスに変換された2件目以降のレコー



索引順編成ファイルの構成

ドをシノニムレコードといい，特別な処理が必要です．

EXAMPLE 1

ファイル編成に関する次の記述中の[　　]に入れるべき適当な字句を解答群の中から選べ．

1．[a]ファイルは，磁気ディスクや磁気テープなどの多くの種類の媒体に作成可能である．一般的には記録した順にしかアクセスできずファイルの更新は原則としてファイル全体をコピーして行う．ただし，[b]は全体のコピーなしに可能である．また，磁気ディスク上に作成された場合には，[c]，や[d]も可能である．

2．[e]ファイルは，主として磁気ディスク上に作成される．このファイルでは，キーを指定すると，対応するレコードの論理的な格納位置が演算によって求められる．

キーが，ファイル中のレコードの番号自身であるときには，とくに，[f]ファイルと呼ばれ，ファイル中にキーを記録しておく必要がない．

キーが，ファイル中のレコード番号自身でないときには，キーからレコードの論理的な格納位置を求めるために，[g]と呼ばれる手法が用いられることがある．

3．[h]ファイルも，主として磁気ディスク上に作成される．このファイルでも，キーにしたがってレコードをアクセスできる．しかし，[e]ファイルの場合とは異なり，キーから対応するレコードの論理的な格納位置を求めるためにキーとレコードへのポインタだけを格納した[i]を設ける．大きなファイルでは，これを階層構造にすることもある．

[h]ファイルのうち，キー順のアクセスを，容易にするように[i]レコードとデータレコードをそれぞれ論理的にキー順に並べたものは，特に[j]ファイルと呼ばれ多く用いられている．

<a,e,f,h,jに関する解答群>

ア．区分編成　イ．索引編成
ウ．索引順編成　エ．順編成
オ．相対編成　カ．直接編成

<b～dに関する解答群>

ア．ファイルの最後部へのレコードの追加
イ．レコードの削除
ウ．レコード長を変更しないレコードの書き換え
エ．レコードの挿入

<g，iに関する解答群>

ア．インデクシング　イ．索引
ウ．ハッシング　エ．ヘッダ
オ．ラベル

昭和59年度　第2種　問4

解説

a 記録した順にアクセスすることを，順次アクセスといいます．順次アクセスしかできないファイルは順次編成です．

b 順次編成ファイルのメインテナンスにおいて，全体のコピーなしにできるのは，ファイルの最後へのレコードの追加です．

c，d 磁気ディスクの各レコードはアドレスを持っているので，特定のレコードに正しく読み書きヘッドを位置づけることができます．そのため，ファイルの更新において，レコードの削除と，レコード長が変らない場合の更新が可能になります．

e 演算によってキーからアドレスを求めるファイルは直接編成です．

f 直接アドレス方式で作成された直接

編成ファイルを，相対編成ファイルと呼ぶことがあります．

gアドレスを取得するための演算のことを，ハッシングといいます．

h，j索引編成か索引順編成のどちらかが答えになります．空欄のjとの関係から，hが索引編成，jが索引順編成となります．

i索引編成において，キーとアドレスを対応づけて記録しているのはインデックス（索引）域です．

解答

a．エ　b．ア　c．イ　d．ウ
e．カ　f．オ　g．ウ　h．イ
i．イ　j．ウ

EXAMPLE 2

ファイル編成法に関する次の記述中の［　　］に入れるべき適当な字句を解答群の中から選べ．

ファイル編成の代表的なものには，［ a ］，［ b ］，［ c ］の3種類がある．

［ a ］は，主としてバッチ処理におけるトランザクションファイルに使用されるファイル編成で，順次アクセスしかできず，記録媒体としては，磁気ディスク，［ d ］などが使用される．なお，磁気ディスクや［ d ］上には，複数のレコードを1ブロックとして記録することが多い．

［ b ］は，直接アクセスできるファイル編成で，記録媒体としては，［ e ］がよく使用される．この編成のファイルをアクセスするには，プログラム中に，レコードの格納場所を指定する［ f ］機能を含んでいなければならない．

［ c ］は，3つの領域，すなわち，オーバーフロー領域，プライム(基本レコード)領域，［ g ］領域より構成される．このファイルを初期作成したとき，プライム領域のレコードは，キー項目の［ h ］順に記録されている．このファイル編成の特徴は，［ g ］領域の情報を使用することにより，直接アクセス，順次アクセス両方を可能にしていることである．

<a,b,cに関する解答群>

ア．関係編成　イ．索引順編成
ウ．順次編成　エ．直接編成
オ．網編成

<d,eに関する解答群>

ア．カード　イ．磁気ディスク
ウ．磁気テープ
エ．ラインプリンタ
オ．OCR

<fに関する解答群>

ア．アドレス生成
イ．ファイル生成
ウ．プログラム生成
エ．レコード生成

<g,hに関する解答群>

ア．昇　イ．降
ウ．インデックス　エ．マスタ
オ．レコード

昭和61年春期　第2種　問5

解説

a順次アクセスしかできないファイルは順次編成です．

bプログラムでレコードの格納位置を決めるファイルは，直接編成です．

c 3つの領域からできているファイルは索引順編成です．

d順次編成を作成できる代表的な記録媒体は，磁気ディスクと磁気テープです．

e直接アクセスができる記録媒体は，磁気ディスクです．

f直接編成ファイルでは，処理プログラムの中でアドレスを作成し，アクセスします．

g索引順編成の3つの領域は，オーバーフロー領域，プライム領域とインデックス（索引）領域です．

hプライム域のレコードは昇順に記録されています．

解答

a．ウ　b．エ　c．イ　d．ウ
e．イ　f．ア　g．ウ　h．ア

プログラム言語

プログラム言語についての出題は，アセンブラ，コンパイラなどの言語プロセッサに関するものと，FORTRAN，COBOLなどの代表的なプログラム言語に関するものの2つにわかれます．

言語プロセッサ

代表的な言語プロセッサ（言語処理プログラム）とその機能を解説します．

■アセンブラ

計算機向き言語の1つであるアセンブリ言語のソースプログラムを翻訳（アセンブル）します．

■コンパイラ

問題向き言語，あるいは，高水準言語と呼ばれるコンパイラ言語のソースプログラムを翻訳(コンパイル)します．

■インタプリタ

プログラムを1命令ずつ解析（翻訳ではありません）しながら実行します．この時のソースプログラムは，アセンブルやコンパイル時のソースプログラムと違い，完成している必要はありません．そのため，プログラム開発が効率よく行えます．

■ジェネレータ

決められた形式にしたがって記述されたパラメータを受け取って，その指示にしたがってオブジェクトプログラムを作成します．

■プリプロセッサ

疑似命令で記述されているソースプ

ログラムや一部の命令を，特定の言語のソースプログラムや命令に変換します．

言語

代表的なプログラム言語とその特徴をまとめておきます．

開発年と開発者は，問題を解く際のキーポイントになります．ただし，開発年は，解釈の違いによる多少のズレがあるので注意してください．

■ FORTRAN

開発年：1956年

開発者：バッカス

現在では広く一般に用いられていますが，もともとは，IBM社のために開発された言語です．手続型言語の中で最も歴史が古く，数式に近い形でプログラムを記述できます．科学技術計算向きです．

■ COBOL

開発年：1960年

開発者：CODASYL委員会

ファイル処理や，データの編集処理のために開発された言語です．数ある言語の中で最も利用度が高く，とくに，事務処理の分野で広く用いられています．

■ ALGOL

開発年：1960年

開発者／開発場所：ヨーロッパ

それまでに開発された言語と違い，構造化の概念をもって設計されました．PASCALなどは，この言語の影響を受けています．科学技術計算向きです．

■ APL

開発年：1962年

開発者：IBM社のアイバーソン

タイムシェアリングのもとで会話的に用いられます．特殊な記号を用いるため，プログラムを簡潔に記述できます．論理演算やマトリックス(行列)演算に大きな力を発揮します．科学技術計算向きです．

■ LISP

開発年：1960年初頭

開発者：マサチューセッツ工科大学のマッカーシー

関数型の言語です．リスト構造（2分木）のデータを処理できるため，人工知能型言語として注目されています．

■ BASIC

開発年：1965年

開発者：ダートマス大学のカーツとケメニー

当初はプログラム教育用としてタイムシェアリングのもとで用いられていました．その後一時期，パーソナルコンピュータ用言語としてパソコンの普及と共に広く用いられました．

■ PL/1

開発年：1965年

開発者：IBM社と，そのユーザ団体であるシェアグループ

FORTRAN, COBOL, ALGOLの各言語の良いところをとって，IBM社のために開発されました．

■ LOGO

開発年：1967年

開発者：マサチューセッツ工科大学のパパート

作図機能が優れています．タートル（亀）というマークを基準に，図形(タートルグラフィックス）を簡単に書くことができます．心理学者ピアジェの理論をコンピュータを通して実証するため，教育用言語として設計されました．リスト構造のデータを処理できるため，人工知能型言語として注目されています．

■ PASCAL

開発年：1971年

開発者：チューリッヒ工科大学のビルト

ALGOLの影響を受けています．プログラム教育を目的として設計され，おもに大学や研究機関で用いられています．C，Adaなどは，この言語の影響を受けています．

■ PROLOG

開発年：1972年

開発者：フランスのマルセーユ大学

3段論法を用いることで，論理的な関係を簡単に表現できるようにした，論理型の言語です．人工知能型言語として注目されています．

■ C

開発年：1970年初頭

開発者：AT&T（米国電信電話会社）のベル研究所

UNIXというOSを記述するために設計された言語です．ハードウェアからの独立を考慮した，移植性の高い言語です．また，開発効率が良いため，現在では多くの分野にわたって利用されています．

■ Ada

開発年：1981年

開発者：米国防総省

エイダ・ラベレース（世界で最初の女性プログラマ）にちなんでこの名がつきました．軍事用に設計されているため，リアルタイム機能が優れています．モジュールごとに作成・翻訳ができ，ソフトウェアの部品化がはかれます．

EXAMPLE 1

プログラム言語と，その処理系に関する次の記述a～eに最も関連の深い字句を解答群の中から選べ．

a 原始（ソース）言語の命令文を，一つずつ翻訳して実行する．

b 原始（ソース）言語の命令文を，まとめて計算機向き言語に翻訳する．

c 計算機向き言語で書かれた原始（ソース）命令文を，対応する計算機言



語の演算コードや記憶域アドレスに書き換える.

d 特定の言語の機能を拡張するために，原始（ソース）命令文を翻訳の前に変換して，許されている命令文に編成する.

e パラメータにしたがって，ある骨組みにしたがった計算機プログラムやジョブ制御文などを作り出す.

<解答群>

ア．アセンブラ　イ．インタプリタ
ウ．コンパイラ　エ．ジェネレータ
オ．シミュレータ
カ．プリプロセッサ
キ．リンケージエディタ
ク．ローダ

昭和62年度　秋期　問7

解説

a インタプリタは，コンパイラのような「翻訳」は行わないので，厳密に考えると，この記述に該当する言語プロセッサは存在しません.

b 本編のコンパイラの説明参照.

c 本編のアセンブラの説明参照.

d 本編のプリプロセッサの説明参照.

e 本編のジェネレータの説明参照.

解答

a．イ　b．ウ　c．ア
d．カ　e．エ

EXAMPLE 2

プログラミング言語に関する次の記述 a ～ e に最も関連の深い字句を，解答群の中から選べ.

a オペレーティングシステム（OS）などの制御プログラムを記述するのに適した言語である．ベル研究所がUNIXを開発するに当たり，この言語を利用している．開発効率が高く，ある機種で作ったプログラムを他の機種に移しかえるのに便利である.

b リレーショナルデータベース（RDB）をアクセスするための言語である．当初は特定のメーカーのRDB専用の言語であったが，今日では多くのRDBで適用可能になっている．1987年にJISとして規格が制定された.

c 初心者向けの会話型言語として，米国のダートマス大学で開発された．1980年代前半にはパーソナルコンピュータ（パソコン）の主要言語であったが，パソコンのOSが進化したことにより，最近は利用度が落ちている.

d 1950年代末に米国のマサチューセッツ工科大学で開発された言語である．プログラムもデータも同じ表現で記述されるのが特徴の一つである．内部データ構造はリスト（この場合2分木）形式で表現される．記号処理分野で広く使われており，人工知能言語と呼ばれることもある.

e 主として科学技術計算向けに作られた言語で，一般に使われる数式に近い形で記述が可能である．計算機の歴史上，最初に実用面で大成功を収めたコンパイラ言語である.

<解答群>

ア．ALGOL　イ．BASIC
ウ．C　エ．COBOL
オ．FORTRAN　カ．LISP
キ．PASCAL　ク．PL/1
ケ．PROLOG　コ．SQL

平成元年　春期　第2種　問8

解説

a 本編のC言語の説明参照.

b SQLは，1982年にIBM社のサンノゼ研究所で開発された言語です．1977年からサンノゼ研究所で稼働しているデータベースシステム「System R」をもとに作成されました.

c 本編のBASICの説明参照.

d 本編のLISPの説明参照.

e 本編のFORTRANの説明参照

解答

a．ウ　b．コ　c．イ　d．カ
e．オ

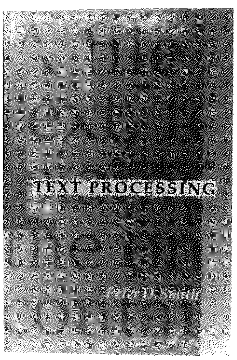
Book Review

文字列マッチングから機械翻訳まで テキスト処理を概説したアンソロジー

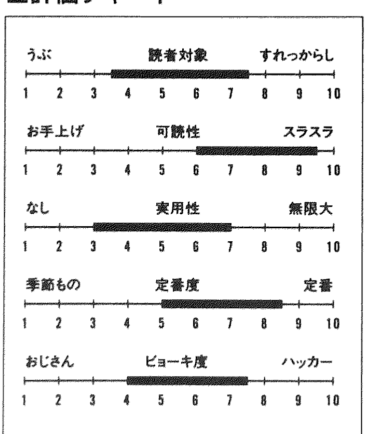
■中村正三郎

■An Introduction to TEXT PROCESSING

Peter D. Smith
MIT Press



■評価チャート



世にアンソロジーというものがある．既存の作品群を別の視点からふるいにかけ，一つの作品集にまとめ上げたものである．恐怖小説集，時代小説集など様々なアンソロジーが出版されている．本書は，さしずめテキスト処理のアンソロジーである．

ワープロの普及を例に出すまでもなく，コンピュータによる処理の中では，テキスト処理は昔から重要な位置を占めてきた．にもかかわらず，テキスト処理は，縁の下の力持ちとして低い位置に置かれがちであり，主役を演じ，表舞台で華々しい脚光を浴びることは少ない．テキスト処理にまつわるアルゴリズムの多くは，基本的なものとして，各分野に分散して埋没してしまうからであろう．

そこで本書の登場となる．テキスト処理という視点に立って，小は文字列マッチングから大は機械翻訳まで，多岐にわたって話題を集めて概説してある．テキスト処理の各分野に全般にわたって概説した書物は，評者の知る限り本書が初めてである．目のつけどころが大したもので，企画の勝利といえよう．

簡単に本書の構成を紹介しておく．全体は5部と付録に分かれる．

I Document Manipulation
II Character - by - Character Processing
III Symbol-by-Symbol Processing
IV Word-by-Word Processing
V Natural Language Processing

Iでは，文書の操作について述べている．文書入力の各種，文書の比較，保管，引き出し（いわゆるIRシステム）といったトピックを取り上げている．

IIからIVは，小さな単位の処理から大きな単位の処理へと構成してある．

IIで扱うのは，エディタ各種（行エディタ，ストリームエディタ，画面エディタ，構文エディタなど），圧縮法，暗号化である．エディタでは，バッファ管理，画面再表示，文字列マッチングのアルゴリズムをいくつか紹介している．圧縮法，暗号化でも同様に技法の紹介がある．

IIIでは，マクロプロセッサ，テキストフォーマッタ，用語索引（コンコーダンス，concordance）などの話題を取り上げている．

IVでは，ハイフネーション，スペルチェックなどの話題になる．

Vは，文章の特徴を調べて著者が誰かを知るAuthorshipの研究，自動要約作成，機械翻訳の話題である．

概説であるため，各トピックについてさほど詳しく解説していないが，参考文献は豊富に挙げられており，後学のためになる．

邦訳の際には，日本語処理にまつわる話題を追加してほしいものである．日本語の入出力，特にかな漢字変換の

技術は，取り上げておかねば日本語テキスト処理のアンソロジーとしては不適当であろう．また，本書で取り上げてある各トピックについても，日本語特有の問題がいろいろとある．それについても指摘してくれれば，ありがたい．

テキスト処理技術の現状を展望し，勘所をつかんでおきたい人は，一度通読しておけば役に立つに違いない．テキスト処理の初心者からベテランまで，薦められる本である．

盛り上がってきたTEXとPostScript関連書

古田島義和（神田・書泉グランデ）

米国で普及している「TEX」が，日本でもようやく動きはじめました．89年ごろから日本語版の解説書が出版されはじめ，代表的な図書にアスキーの「TEXブック」がありますが，これは，Addison Wesley（以下AW）の「The TEX BOOK」の訳本でもあり，いうまでもなく米国スタンフォード大学のD. E. Knuth氏の著作でもあります．このTEXは文書処理システムとして開発されたソフトで，通常のDTPソフトや一太郎などのワープロと違い，文章の中に文字の大きさや種類・行間などを指定するコマンドを入れて，一括処理する便利なソフトです．

日本語で使えるTEXも，アスキーによって改良され「日本語TEX」として発売されました．このTEXから派生した文書整形システムが「LaTEX」で，AWから「LaTEX」（Leslie Lamport著）が出ています．訳本（タイトルは同じ）は，アスキーから出ており，最もよく売れているもののひとつで，バイブル的な存在になっています．今年4月に出版された秀和システムの「LaTEXトータルガイド」は，この盛り上がりの中で，ベスト17位と健闘しています．

ほかにPostScript関連書がよく売れています．これは，米国Adbe Systems社の商品で，文字や図形，ページレイアウトを作画するためのプログラミング言語で，AWから3分冊「Reference Manual」「Tutorial and Cookbook」「Program Design」が出版されており，いずれもアスキーから訳本が出ています．この中で「PostScript Language Reference Manual」が，AWから改訂第2版が出版され，今月のベスト10入りしています．これらのさまざまなソフトは出版・印刷などに大きな影響を与えることでしょう．

今月の洋書新刊リストを紹介したいと思います．OSやネットワークで有名なComer氏とベスト10の上位に常にいる「UNIX Network Programming」の著者Stevens氏の共著で，Prentice Hall（以下PH）から「Internetworking with TCP/IPII」が出版されます．前回のIはComer氏のみの作品でしたが，前回を上回る人気は確実でしょう．PH「Compiler Design in C」，Holub著の第2弾「Compiler Design in C Software」，AW「The Design and Implementation of the 4.3 BSD UNIX Operating System」（6月か7月に丸善より訳本が出版される予定），Leffler著「Answer Book」が出版されたり，ほかにAW「The C++ Programming Language」（Stroustrap著），AW「A C++ Primer」（Lippman著）が第2版にかわるなど，内容が一新される図書が続々と登場します．

最後に注目されている分野の新刊案内をしたいと思います．AW「OSF/Motif:Concepts and Programming」（Berlage著），AW「Visual Design with OSF/Motif」（Kobara著），AW「TCP/IP and NFS:Internetworking in a UNIX Environment」（Santifaller著），AW「Distributed Window Systems:A Practical Guide to XII and News」（Davison著），PH「Introduction to the Theory of programming Languages」（Meyer著）．この同氏の作った言語の解説書としてPHから「Eiffel:The Language」が出版．第一線の著者の出版がめじろ押しと，楽しみな月となるでしょう．

■和書&洋書ベスト10

（書泉グランデ調べ：1991年4月1日～4月30日）

1	VZ倶楽部	ビレッジセンター
2	SUNシステム管理	アスキー
3	C言語による最新アルゴリズム事典	技術評論社
4	続ソフトウェア千夜一夜物語	共立出版
5	入門C言語　改訂新版	アスキー
6	コンサルタントの秘密	共立出版
7	たのしいUNIX	アスキー
8	XウィンドウとUNIX・GUI	ソフトリサーチ・C
9	ノベルのNetWare戦略	オーム社
10	rootから/へのメッセージ	アスキー

1	Undocumented DOS	Addison Wesley
2	ResEdit Complete	Addison Wesley
3	Compilers	Addison Wesley
4	Simple Book:An Introduction to Management of TCP/IP	Prentice Hall
5	Object-Oriented Analysis 2/E	Prentice Hall
6	System performance Tuning	O'Reilly
7	UNIX Network Programming	Prentice Hall
8	Programming Windows 2/E	Microsoft Press
9	LAN Protocol Hand Book	Prentice Hall
10	PostScript Language Reference Manual 2/E	Addison Wesley

◆ ◆ 佐々木哲のニューヨーク ◆ ◆

N.Y. システム通信

帝国の逆襲?

IBM PCは偉大な標準です.

古の日々はIBMがハードウェアを作り，その仕様を公開する.それを元に多くのソフトウェア,ハードウェアメーカーがIBM PCの上で動く製品を開発する．さらには，IBM PC自体と互換性をもつコンピュータを作るという構図でした．いくら安価な高性能のハードウェアが出てこようと，マーケットのスタンダードはIBMでした.

そのうちに主従がはっきりしなくなりました．IBM純正品のシェアが落ち，またIBM自身がPS/2を紹介することにより，IBM PC/AT互換を捨てたために，標準を提供する地位から落ちてしまいました．PS/2シリーズなどいい例です.ソフトを書く側にとっては,単なる互換性がすこし欠けるAT互換機です．ハードウェアだけでなく，OS/2の不調により，OSさえもコントロールに失敗しています.

ここのところIBMの巻き返しともいえる動きがめだちます．再び魅力のある製品を出すことにより大勢を巻き込む．たとえばOS/2 2.0．これは，Microsoftの Windows 3.0への対抗心を見せています．Windows 3.0のいささか不安定なDOSアプリケーションのサポートとは異なり，16個までのDOSアプリケーションを安定して動かし，Windowsのアプリケーションも動作可能.さらには,互換機での動作まで検証する．いま現在，これのベータテストが進められています．すでに，IBM内部では10,000近いベータユーザーがいるようですが，これを一般に広めます.

極端な例は，Atlantaにある全米唯一のIBM公認ユーザーBBS*1でのOS/2 2.0ベータコピーの配付です．10 MB近いファイルをここで無審査で配付しています．ライセンス条件の関係から米国内のユーザーに限られていますし,再配付はかたく禁じられています(この原稿を書いた時点では一時的に配付は中止されています).このような過激ともいえる動きが見られる最近のIBMです.

＊1:IBM National Support Center BBS
BBS (404) 835-6600
300/1200/2400/9600 8/N/1

もっと大きく，速い表示を!

先月でも少し触れたように，VGAの次のディスプレイの標準は混沌とした状況です．IBM自身は8514/Aを捨てXGAに移りましたが,最初に述べたように,すでにマーケットをコントロールできる地位にはありません.なんとかXGAを標準にすべく,IBMはノンインターレースのXGAの開発，ISAバス用のXGAの開発を表明しています．また,他社へのチップセット，技術の提供を図っています.

一方,ディスプレイボードメーカーは,最初仕様が公開されなかった8514/Aを3年がかりで解析，クローンし,これから販売というときに,IBM自身が8514/Aを捨てたことにより困惑しています．とはいえ，8514/A互換ボードは，先月のATIの高性能ATI 8514/ULTRAをはじめいくつも安価に提供されています.

さらには，Super VGAのほうはとどまるところを知らない発展を示しています.1024×768をこえたボードもでてきました．高性能アドオンCPUボードで有名なDefiniconのHiRes 1280*2です.これは,1280×1024の解像度で16色あるいは1280×960で256色の表示ができます．例によって，Windows 3.0，AutoCAD，Lotus 1-2-3，Ventura Publisher，WordPerfectなどのドライバがついてきます．メモリは2MB搭載.

2＊:HiRes 1280 $695
Definicon International Corp., 17312 Eastman St., Irvine, CA 92714, phone (714) 961-0438.

WeitekのGraphics Coprocessor

高性能の数値演算チップで知られるWeitekも，このマーケットに参入しました.

WeitekのUser Interface Controllerという名前のチップは，ISAバスを対象にSuper VGAにGraphic Coprocessorの機能を付け加えたものです．MS－Windowsに最適化された機能をもっています.

専用のドライバにより，Windows GDIのBitBLTおよびline drawingのみをハードウェアで実行するものです.Weitekによると,MS-Windowsのオーバーヘッドの

85%はBitBLTとline drawingであることが，MS-Windowsアプリケーションの動作の解析により明らかになったとか．Super VGAなどでは，これをすべて本体CPUが実行しているため，MS-Windowsの動作が遅いものになります．これを専用ハードウェアに任せることにより，最小の手間で全体の高速化を図ります．IBMのXGAも同じ視点で高速化をはたしているのでしょうが，価格の点で勝負になりません．

解像度は，ノンインターレースで最高1024×768の16色までです．これを利用したボードは，いまのSuper VGAボード程度の値段で，今年末には各社から現れてくることでしょう．

IBMの新VGAチップ

IBMのPS/2シリーズは，最初のものからすべてVGAを本体に含んでいます．ただし，何の仕掛もない8ビットバスのVGAです．Tseng ET4000など，多くの高速なSuper VGAチップセットの登場とともにいささか見劣りがしてきています．PS/2のグラフィックはとても遅いのです．

XGAでハイエンドを整備しつつあるIBMですが，この問題に対処するため新たなVGAチップを開発し，ローエンドのPS/2ラインに搭載をはじめています．その最初が，日本で発売されたPS/55の5510Tモデルです．このチップセットに関しては，800×600の解像度をもつという話もありますが，IBMは否定しているようです．XGAの開発の結果がフィードバックされ，とても高速なVGAとなっています．

実際の計測によると，Micro Channelバスの高速性を生かし，ISAバスでの最高速チップであるTseng ET4000をこえる速さを見せています．インテリジェントではないVGAの場合，性能は表示メモリへのアクセスの速さで決ります．すでにISA 16ビットバスの限界にまで達している現在のSuper VGAに十分対抗できるものです．PS/2ラインがモデルを一新するまでは，ローエンドの新しいモデルのほうがPS/2 model 70などより表示が高速という状況が見られることでしょう．なお，この新VGAチップにはいくつもの付加機能があるようですが，互換メーカーを牽制するため新レジスタ類は公表されないようです．

さてさて，このグラフィック標準を巡るバトル，どうなることやら．しばらくは目が離せません．来月はIBMがまったく手をつけていないVGAの多色化についての動きにふれます．

今月のなんだこりゃ？

今回はハードウェアの話ばかりになってしまいました．やはりこの豊富で安価なハードウェア環境がIBM互換機の世界の魅力です．

以前，499ドルの9600bps V.32モデムを紹介しました．Practical Periferal，ATIどちらも定評のある会社です．今度は，安価というよりcheapなFAX/modemで知られる会社からのV.32モデムです．

Image Communicationsは，なんと299ドルの内蔵V.32モデムTwinCom 96/42*3を発表しました．もちろん定価です．このモデムは，V.32にV.42，V.42bis，MNP4のエラー訂正機能をもっています．普通のディーラーを通してのほかに，直接メールオーダーでも販売するそうです．

どうやったら，これだけ安くできるのか．その秘密はRockwell 9696ACというモデムチップにあるようです．上記のスペックは，ほとんどがこのチップに含まれています．ただ単に，このチップをボードにまとめただけというものでしょう．あとはコストを下げるためにサポートはなし，トラブルがあったら即交換という方法をとるようです．

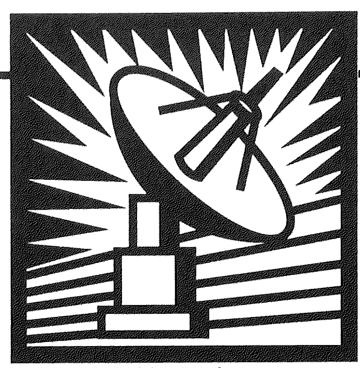
*3:TwinCom 96/42 $299.00
Image Communications Inc., 6 Caesar Place Moohachie, NJ 07074, phone (201) 935-8880



Illustration : Masami Nozue

SYSTEM N.Y. NEWS

Illustration : NOBUKI FUKUHARA



西川和久の
希望的観測

1回休憩!? 「話題がない」という話題について

1 国内外でビジネスショウ，COMDEX 5月は業界イベント真っ盛り

5月は，いろいろなパソコン業界のイベントが，国内外問わず行われている．国内で最大規模のものは「ビジネスショウ」，国外では今日（5月20日）からはじまる(^^;)「COMDEX Spring」であろうか．編集者の方から「早く原稿を……」といわれながら，「ビジネスショウが終われば，いろいろなネタがでてくるので，もう少し待って……」と苦し紛れにいいわけしていた私にとって，今回のビジネスショウはまったくおもしろみのない内容であった．すなわち，ネタになるようなものがなかったのだ…….

とはいえ，原稿締切のデッドエンドが間近なために，次の内容を考えている暇もない．当初の予定通り，今回は私が足を運んだ2つのイベントについて紹介する．

2 「日本IBMはかわった」 盛況だったDOS/V Power Forum

最近，いろいろな雑誌などで騒がれているDOS/Vのプライベートフェアが5月10日(金)～12日(日)に，北青山TEPIAエクジビジョンホールで行われた．通常，日本IBMのイベントは日曜日を含まないのだが，一般大衆を狙ってこの日程にしたと考えられる．実際，金曜日は業界人の山だったが，土日に関してはTERADRIVE目当ての親子づれなどが多く見られた．DOS/V関係のイベントとしては，はじめてということもあってか，3日間で約6,000もの人が集まったようである．

内容としては，直前で発表された5510Zや5510T，すでにお馴染みのPS/55noteやセガのTERADRIVEなど，DOS/Vマシンと対応ソフトがところ狭しと展示されていた．なかでも目を引いたのは，米国で次世代の画面規格と呼ばれているXGAのデモであった．5510Tに搭載されたXGA上で「蝶が飛び回る動画」や，驚くことに「XGA日本語Windows3.0」までもが参考出展されていたのだ．また，セミナーや対談などは，毎度お馴染みの面々や芸能人を起用し，内容もなかなか面白いものであった．「DOS/Vや日本語Windowsに関しては日本IBMは変わったな……」というのが私の率直な意見である．

大阪でも6月に同フォーラムが行われるが，どれだけの人を集めるか，いい方をかえれば，東京以外でどれだけDOS/Vに関心をもっている人がいるか？　非常にたのしみである．

3 半年前から騒がれているものばかり 新鮮味に欠けたビジネスショウ'91

5月15日(水)～18日(土)まで，晴海の国際見本市会場で行われたビジネスショウ'91，前評判としては日本語Windows 3.0，DOS/V，NetWareなどいろいろあったが，実際のところあまり新鮮味のある内容ではなかった．これはたぶん，各アイテムがすでに半年近く前から騒がれているものばかりであるためだろう．展示の仕

ワンラインNEWS
ナムコは業務用ゲーム機器の国内販売を強化した．これまでは東京と大阪にある事業所で全国をカバーしてきたが，新たに5か所の販売拠点を開設した．

方やデモなども，とくに目新しさはなかった．

パソコン関係各メーカーの新製品についても今回は不作（？）で，NECのニューモデル，AX-VGAマシン，SONYのPalmTop，DOS/Vマシン程度であった．個人的に面白かったのは，日本IBMブースで見た「OS/2 J2.0」，それに話題のLotusとジャストシステムの合作，「HARMONY」．もし，今回この連載でとりあげるとしたら，この2つのことについて書いていたであろう．

ソフトウェア展示が主のソフトウェア館では，年々有名な会社のブースがなくなっている．不思議に思いある関係者に理由を聞いたところ，「お金がかかる割には効果があがらず，既存ユーザーのQ&Aコーナー化してしまう」ためだそうだ．

ここ何年かのビジネスショウは，昔と違って数社のメーカーに人が集中してしまう傾向がなくなってきている．集客力という面では，どこのメーカーも平均化しているのだ．これは非常によいことなのだが，反面，面白みに欠けてしまうのも事実である．会場を歩いている人がもつ紙袋が，特定のメーカーでしかなかったころを懐かしいと思っているのは私だけではないと思う……．

4 COMDEX Springもはじまるけれど……毎回テーマ探しはたいへんだ

世界規模で最大のパソコンショウが今日からはじまる．COMDEXである．春と秋に2回開催されるが，残念ながら現時点ではまだ情報は入ってこない．たぶん，次号あたりで隣の「佐々木哲のニューヨークシステム通信」で何か面白い情報が登場することでしょう……．

＊＊＊

いいわけになってしまうが，毎月この連載の表題を決めるには非常に苦労する．私が興味をもち，かつ業界や読者の方にインパクトがあるようなネタは，そう毎月発生するわけではないのだ．毎号，編集者の方といろいろネタについて話合い，決まるまでに非常に時間がかかってしまう．いったん，書きはじめるとほんの2～3時間で終わるのだが，そこに至るまでがたいへんなのだ．また，話が多少（？）IBM PC系の内容に片寄ってしまうのは，個人的な趣味であるからお許しいただきたい．%-)



Computer Laws

コンピュータの法律

プログラムの改造・その1

熊谷信太郎(弁護士)

■プログラム改造の問題点

今回から数回にわたって，プログラムの改造に関する諸問題をとりあげたいと思います．

プログラムは著作権法上，著作物として保護されているわけですが，プログラムの改造については，

(1) 同一性保持権との関係
(2) 翻案権との関係

がそれぞれ問題となります．

このうち，同一性保持権とは，著作権者に認められている著作者人格権の一種です*1．すでに説明したように，著作者は，「著作者人格権」と個別の財産権としての「著作権」の2種類の権利を有しています．一般にはその両者を併せて「著作権」と呼ぶ例もありますが，厳密には，両者は別のものなのです．

■同一性保持権の制限

著作者人格権は，同一性保持権，公表権，氏名表示権の三つで構成されています*2．同一性保持権とは，著作者の意に反して著作物を勝手に変更，切除その他の改変を受けない権利です．

実は，日本においてプログラムを著作物として保護すべきか問題になった際に，それに反対する立場からは，もし，プログラムが著作物ならば，プログラム製作者には著作者人格権の内容をなす，同一性保持権が認められることになるが，それは大変不都合な結果になるから，認めるべきでないと主張されていたのです．

すなわち，プログラムのように，バージョンアップすることが必要な著作物について，同一性保持権を主張されると，バージョンアップができなくなるおそれがあるし，また，自分のコンピュータで使えないプログラムを使えるように改造することも制限されてしまい，不便きわまりないというのです．

このような，反対論の主張にもうなずけるものがありましたが，一方で，コンピュータの発達にともなうプログラムの法的保護も必要でしたので，結局，著作権法は，プログラムを著作物として，著作権法で保護することとし，同時に，反対論があげる不都合については，立法で特別な手当てをすることで解決したのです．

それが，現在の著作権法に置かれているプログラム著作権者の同一性保持権の制限なのです．そこでは，「特定の電子計算機においては利用しえないプログラムの著作物を当該電子計算機において利用しうるようにするため，又はプログラムの著作物を電子計算機においてより効果的に利用しうるようにするために必要な改変」については，プログラム著作物の同一性保持権は，及ばないとされています*3．

■より効果的に利用するための改造

このうち，「特定の電子計算機においては利用しえないプログラムの著作物を当該電子計算機において利用しうるようにするため」というのは，コンピュータを作動させるための小規模な改造を意味するものと解されており，プログラム自体が不十分，あるいは未完成なために，プログラムの同一性を害するような大改造をしなければならないような場合は含みません．

また，「プログラムの著作物を電子計算機においてより効果的に利用しうるために必要な改変」とは，いわゆるバージョンアップを意味します．

このように，「より効果的に利用」するための改造が認められたため，プログラムについては，同一性保持権に関する制限がほとんどなくなり，プログラムの改造が大幅に認められることになったのです．

もっとも，最近のこの点に関する議論としては，市販のプログラムを買う際の売買契約に，改造禁止の特約がついている場合には，この著作権法の規定と，その特約のどちらが優先するのかという問題があり，両説が対立しています．

また，注意すべきは，プログラムの改造は，「著作者人格権」としての

同一性保持権と，個別の財産権としての「著作権」の双方で規制をしていますから，そちらの検討もしなければ，プログラムの改造が許されるかどうかを論ずることはできません．

これらの問題については，次回以降に検討したいと思います．

＊1　著作権法第20条第１項
＊2　著作権法第18条，第19条，第20条
＊3　著作権法第20条第２項第３号

アキバ短信

秋月電子

週末には元ラジオ少年達がたむろする伝説の店

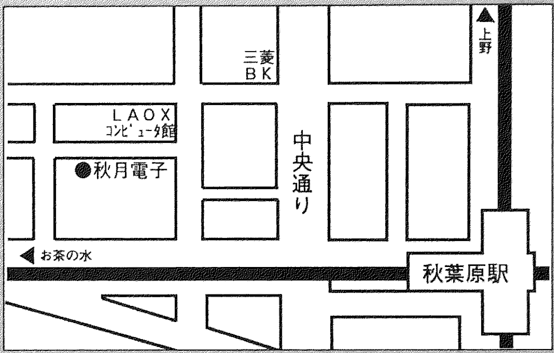
秋葉原には俗に「ジャンク屋」と呼ばれる店がある．ジャンク（junk）とは，本来ガラクタとか屑ものという意味だが，もちろんゴミを売る店ではない．狭いスペースに汚い部品をぎっしり詰め込んで，どう見てもゴミためにしか見えない店もあることはたしかだが，実際に取り扱っているのは中古，あるいはメーカーの在庫処分品（したがって新品）の電気製品や各種の部品などである．

秋月電子は，ジャンク漁りをしようという人なら誰もが知っている超有名店のひとつだ．かつて信越といっていたころは，日曜日には開店時刻前に店の前に行列ができたとか，新幹線が運転をストップするとその日は休業になるとか，さまざまな伝説(？)が生まれた店なのである．「ああ，アキヅキ（または信越）ね」と思った貴方はここから先は笑って読み飛ばしてほしい．だが，秋葉原でパソコンソフトと家電しか買ったことがないという方はもう少しおつきあい願おう．

秋月電子はおもにIC関係を取り扱っている．特徴はICを各種キットとしてセット販売していることだろう．たとえば「超音波デジタル距離計キット」「デジタル温度計キット」「８色バトルサウンドキット」「ガイガーカウンタキット」(！)といった具合だ．「へぇ，そんなものいったい何に



営業時間　11：30〜18：30（日曜は18:00まで）
月・木定休（祭日と重なる場合は営業）
TEL 03-3251-1779



使うの？」と思う人ははじめからここにはこない．この店の常連を自称する某氏よれば，秋月にくる客は２種類にわけられるという．第１のパターンは，ある特定の部品を買いに秋葉原へきたのだが，特別に安く買えないかどうかを見るという人．第２のパターンは，あてもなくここへきて，何かないかなぁと部品をながめている人．安い部品を見ると突然「これは××に使えるぞ！」とインスピレーションがひらめくことがあるのだそうだ．いずれにせよ，客は自分だけの「掘り出し物」を求めて，黙々と部品をながめる．土，日はこういう人々で店先があふれかえる．ある女性は，秋月電子の前に集まる人々はみな同じような服装であるという鋭い指摘をしている．つまりシャツにGパン（これにジャンパーが加わることも多い），ショルダーバッグである．

店員さんに「この部品は何に使うんですかぁ」などと質問するのはマナー違反というものだ．売っているキットが何の役にたつのか，その部品をどうやって使うのか，それはひたすら買う人それぞれのもつ技術に委ねられている．ジャンク漁りは選択眼と技量だけが頼りのキビしい世界なのだ．

秋月電子の前の通りはほかにも丹青通商，湘南通商，千石電商，ADOパーツショップ，鈴商などジャンクやパーツの店が多いので，ぶらぶらと歩いてみるだけでもその独特の雰囲気は十分に味わってもらえると思う．

●プログラマの仕事場 | 4

株式会社バックス

ポータビリティの高いソースを書くという基本姿勢がVJEのマルチOS対応を可能にしている

いまや，日本語入力フロントプロセッサは，パソコンを使う人なら誰もが知っているソフトウェアになった．この「かな漢字変換機能」をMS-DOS環境のツールとして，はじめて開発・販売したのがバックスだ．MS-DOS，OS/2，Windows，UNIXなど，さまざまなOSに対応した製品を次々とユーザーに提供する同社の開発環境と今後の展開について，社長の萩原健氏，取締役営業企画担当部長の新井俊彦氏にお話をうかがった．

MS-DOSの移植がきっかけでパソコンソフトの世界に

バックスが設立された1981年当時は，まだパソコンを専門にしたソフトウェア会社はほとんどなかったという．バックスも最初はシステムハウスとして，インテル系のCPUを利用したハードウェアを作成し，それにソフトウェアをのせて出荷していた．当時はCP/MなどのOSを利用していた．

その年に松下通信工業からの依頼で，MS-DOSの移植を手がけることになる．これが，パソコンの世界と大きな関わりをもつきっかけとなった．言語はアセンブラが中心だったが，82年ごろからC言語を使いはじめる．そのころCの処理系といえば，Lattice Cのことだった．使えるかどうか実際に試させてほしいと，発売元のLIFEBOATに直接ソースプログラムをもち込んで，その場でコンパイルしたこともあるという．

「システムハウスというと，どうしてもハードウェアの在庫を抱えなければなりません．そこで，MS-DOSの移植をはじめたころから，徐々にソフトウェアの比重を高め，BIOSやデバイスドライバ，LSIを直接コントロールするファームウェアといったシステムプログラムを作成するようになりました」と萩原社長は語る．

「MS-DOS移植の産物」の日本語入力FEP，いまでは主力製品に

設立当初は，ソフトウェアは受託開発が100パーセントであったが，現在では売上の約90パーセントが自社製品．その中でも大きな比重を占めるのが，日本語入力フロントプロセッサVJE-βとVJE-γである．デバイスドライバという形式で，日本語入力の仕組みをMS-DOSに付加するというアイデアは，同社がMS-DOSの移植に携わっていたことと，深い関わりがあるという．

「MS-DOSの移植，日本語化は基本的に日本のマイクロソフトさんがやるわけですが，かな漢字変換などはMS-DOSのOEM供給を受けたパソコンメーカーさんが，それぞれ独自につけ加えることになっています．われわれはそのお手伝いをしていました．DOS上でも何とか日本語入力ができないだろうかと考えて，MS-DOSのVer. 1.25のころから常駐型のかな漢字変換ソフトのようなものを考えていました．Ver. 2.11にな

って，デバイスドライバという考え方が出てきたので，これを使ってできるのではないかと思ったのです」と萩原氏．こうした経緯で生まれたのが，最初の日本語入力フロントプロセッサ「VJE-86」である．これは，MS-DOS普及期の日本語入力ツールとして重要な役割をはたした．VJE-86は，そのあとVJE-α，VJE-βへとバージョンアップされ，単体で販売されるほか，数多くのアプリケーションソフトにバンドルされることになる．

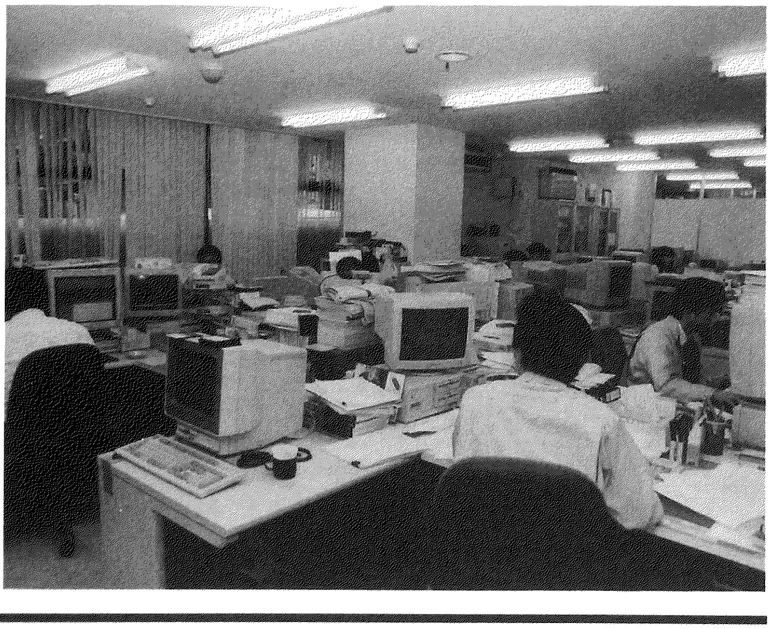
開発はC言語中心で はじめから移植を意識して設計

現在社員は22名．そのうち約4分の3が開発要員，残り4分の1が営業である．

プログラミング言語はほとんどがC．Cで書けない部分はアセンブラを使うが，「UNIX，OS/2，MS-DOSではもちろん，お客さまの要望次第ではいろんなOS——OS-9000，BTRONなども使います．いまではどのOSにも，何らかのCの処理系がありますからね．Cすらない環境にほかの言語で移植するということはまずないと思います．前に1度だけ，FORTRANでというお話がありましたが，うちではサポートしきれないので，ソースライセンスというかたちで出しました」と新井氏．

現在では，どのOSでも使えるCのメリットを生かし，さまざまなOSに対応できるよう，プログラムの設計段階から移植を意識して開発しているそうだ．

また，最近ではNeXTのプログラムを手がける関係から，Objective-Cの勉強もはじめている．「私たちにとって言語というのは手段にすぎません．そのOSにはその言語しかないといわれればそれを使います．基本的な考え方として，言語仕様のうち高度なものは利用せずに，できるだけポータビリティの高いソースを書くようにしています．そういった意味では，Objective-Cは必ずしもポータビリティの高い言語ではありません．しかし，開発するベースとしてのインターフェースビルダー，オブジェクト指向言語としてのObjective-Cというのは，プログラマとしては知らなくてはなら



About Company

萩原健社長．1981年東京都町田市でシステムハウスとして設立．設立当初は各種OSの移植や日本語化などの受託が中心だった．大手メーカーからの依頼で日本で最初にMS-DOSの移植に着手．現在では「VJE-β」，「VJE-Pen」などの自社製品が売上の約90パーセントを占める．



萩原社長（写真右）と新井取締役部長

Location

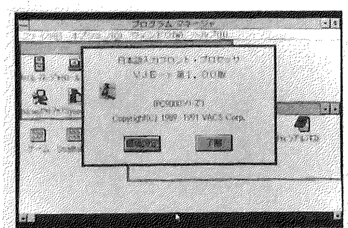
東京都町田市森野．町田までは小田急線急行で新宿から約40分ほど．バックスのオフィスは駅から徒歩5分のビルの1階．ウィンドウに「VJE-β」など同社製品のパッケージが展示されていて，一見しただけではここでソフトウェアを開発しているとは思えない．

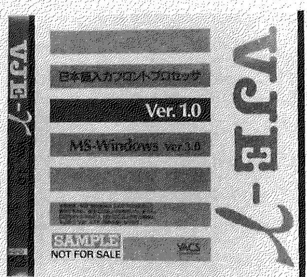
Products

「VJE-β Ver.2.5」（18,000円）
日本語入力フロントプロセッサ．EMS，XMSに対応し，EMS利用時には常駐サイズが約10KB，EMS・XMSを使うと300バイトまで縮小できる．辞書には約7万語が登録され，さらにユーザー登録も約5千語まで可能．オプションとして，キーボードマクロ，10進・16進電卓，ファイルへの切り貼りなどの機能がある．NEC PC-9800シリーズ用，IBM PS/55シリーズ用のほか，UNIX System/V用の「VJE-βストリーム版」などがある．辞書ファイルを編集するための各種ユーティリティを集めた「VJE-Tools」（12,800円）も別売されている．

「VJE-γ」
（Windows用18,000円，OS/2用38,000円，SONY NEWS-OS用63,000円）
日本語入力フロントプロセッサ．「VJE-β」

との違いは変換のロジックとマルチコンテキスト対応であること．マルチタスク用のVJEというわけだ．NEC PC-9800シリーズのMS-Windows3.0対応版（VJE-γ Ver.1.0），MS OS/2 Ver.1.21対応版（VJE-γ Ver.2.2），SONY NEWS-OS Release 2.2/3.2対応版（VJE-γ Ver.1.1）などがある．





「VJE-Pen」（35,000円）
VJE-β Ver.2.5を標準添付した日本語ワープロソフト．操作性を重視した設計で，高速な画面スクロールや[CTRL]キーを使った操作方法などはスクリーンエディタ並．文書のオートセーブ機能や英文ワードプロセッサ機能など，ほかのワープロには見られない独自の機能を備える．

Objective-C

Smalltalkのようなメッセージ送信の機構を取り入れ，オブジェクト指向の考え方をCで実現しようとする言語．NeXTのウィンドウシステムであるNextStepでは，システム記述言語として採用されている．

ないものになるでしょう．そういった意味で，いっしょうけんめい勉強しています」と萩原氏．

開発環境は意外にシンプル VJE-PenとSYMDEBが活躍

気になる開発環境だが，NECのPC-9800シリーズかEPSONのマシンが1人に1台．これがEthernetで結ばれている．このほかに，開発の対象となるOSが動作するマシンが数台，都合ひとりあたり2～5台のマシンをもつ．それぞれ，自分の開発するOSによってメインの開発マシンは異なる．

ファイルサーバーには，製品のマスターの履歴，バグフィックスの履歴，ソースコードなどがのっている．現在は，ネットワークOSにMS-Networksを使用しているため，クライアント同士でメールを出し合ったりはできない．そこで，新しいライブラリやバイナリができると，黒板に書いたり，大きな声で知らせたりと原始的な手段で告知している．今後は，LANマネージャなどのネットワークOSを導入したいとのことだ．

パソコンでのCの処理系はMS-Cを使っているが，PWBやCodeViewは使わない．エディットするときは自社製品のVJE-Pen，デバッグにはSYMDEBを使うか，さもなければICEのお世話になるという．対象となるプログラムがデバイスドライバなどが多いという特殊な事情もあるようだ．「SYMDEBからいっきにICEまでいってしまう．中間がないんですよね．デバッギングツールのいいものがないか探してはいるんですが」と新井氏は話す．

DTPやマルチメディアにも進出 VJEで広辞苑がひける日も近い？

さて，すでにVJEで確固たる実績のあるバックスだが，今後はどのような方面に展開していくのだろうか？　新井氏は，次のように話してくれた．

「まず，考えているのはWindows3.0上で動くレイアウターです．これは，個人でも使えるコストパフォーマンスのよいものにしたいと考えています．VJE-Penは，文字飾りなどの印刷まわりが弱いですから，それを補強する意味もあります．VJE-Penのスピードや操作性を犠牲にして機能を強化するより，別のソフトとして提供したほうがいいと思うんです．

そのほかに，ビジネスショウではNeXTのブースで，PenとかCD-ROMとかβを新しいかたちにしたものをだそうとしています．以前からあちこちでお話していたのですが，CD-ROMを使って英和，和英，類語の辞書を引けるようにしようというものです．それをどうやってMS-DOS上にもってきて，いまのVJEと一緒にするかというのはまだこれからの話なのですが，将来的にはぜひとも実現したいと考えています．ただ，CD-ROMのドライブはオーディオのものにくらべるとまだ値段が高いですね．CD-ROMの普及のためにも，もっと値段を下げてほしいです」．

日本語入力フロントプロセッサVJEで幅広い信頼を得ているバックス，これからも日本語処理にこだわりをもちつづけていくようだ．

information・page

イベント

■7月9日(火)～7月11日(木)
AI'91
AI、ファジー、ニューラルネットワークに関する最新のハード・ソフトを展示。
リスク&セキュリティ・マネジメント'91
高度情報ネットワークにおける安全性・信頼性を確保するためのハード・ソフトを展示。
会場：東京流通センター第1・第2展示場(平和島)／入場料：1,000円(2展共通)
問い合わせ先：日本経済新聞社事業局 TEL 03-3256-5746

■7月9日(火)～7月12日(金)
インターオプト'91
オプトエレクトロニクス関連の素材から応用システムまで最新の光産業技術を展示。
会場：日本コンベンションセンター(幕張メッセ)／入場料：無料
問い合わせ先：(財)光産業技術振興協会インターオプト事務局 TEL 03-3508-2091

■7月10日(水)～7月11日(木)
第17回jusUNIXシンポジウム
UNIX関連の技術論文の発表会。
会場：農林年金会館(東京)
参加費：5,000円(会員)、10,000円(非会員)、3,000円(学生)
問い合わせ先：日本UNIXユーザ会 TEL 03-3356-0156

■7月24日(水)～7月26日(金)
'91教育総合展
教育関連のハード・ソフトの展示。セミナーも開催される。
会場：科学技術館(東京)／入場料：500円
問い合わせ先：日本教育新聞社 TEL 03-3461-7734

セミナー

■6月24日(月)
32ビット対応DOSエクステンダーセミナー
従来の80286ベースのDOS/16Mと比較しながら
32ビット対応バージョンの最新技術を提供。
会場：YMCAホテル(東京)／受講料：20,600円
問い合わせ先：(株)ライフボート
LB ONE DAYセミナー総合案内 TEL 03-3293-4716

■7月5日(金)
C言語アプリケーション構築セミナー
PP IIを使ってポップアップメニューなどのインターフェースの構築方法を解説。
会場：YMCAホテル(東京)／受講料：20,600円
問い合わせ先：(株)ライフボート
LB ONE DAYセミナー総合案内 TEL 03-3293-4716

Microsoft University

■6月24日(月)～6月28日(金)、7月29日(月)～8月2日(金)
MS OS/2 Presentation Manager Applications Programming
OS/2 PMアプリケーション開発のための上級コース。
会場：三田会場、特別会場T／受講料：150,000円(5日間)

■6月25日(火)～6月28日(金)、7月23日(火)～7月26日(金)
Microsoft LAN Manager Programming Environment
LANマネージャアプリケーション開発のための入門コース。
会場：特別会場T／受講料：150,000円(4日間)

■7月1日(月)～7月5日(金)、7月15日(月)～7月19日(金)
8月5日(月)～8月9日(金)
Microsoft Windows Programming Environment
Windowsアプリケーション開発のための入門コース。
会場：三田会場、特別会場T／受講料：150,000円(5日間)

■7月8日(月)～7月12日(金)、
7月29日(月)～8月2日(金)
Microsoft Windows Applications Programming
Windowsアプリケーション開発のための上級コース。
会場：三田会場／受講料：180,000円(5日間)

■7月8日(月)～7月12日(金)
Microsoft LAN Manager Applications Programming
LANマネージャアプリケーション開発のための上級コース。
会場：特別会場T／受講料：180,000円(5日間)

■7月15日(月)～7月19日(金)
MS OS/2 Presentation Manager Programming Environment
OS/2 PMアプリケーション開発のための基本コース。
会場：三田会場／受講料：150,000円(5日間)

■7月23日(火)～7月24日(水)
C言語入門
QuickCを使ったC言語の入門コース。
会場：特別会場T／受講料：60,000円(2日間)

■8月5日(月)～8月9日(金)
MS OS/2 Applications Programming
OS/2 APIを使った実習中心のコース。
会場：三田会場／受講料：150,000円(5日間)

■8月7日(水)～8月9日(金)
C言語応用
MS-C 6.0を使ってC言語の高度な技術を学ぶ実習中心のコース。
会場：特別会場T／受講料：90,000円(3日間)

Microsoft Universityは、OS/2、Windowsアプリケーション開発者のために
定期的に開催されているセミナーです。
問い合わせ先：富士ソフトウェア(株)MU教育センター TEL 03-3452-

NEWS & NEW PRODUCTS

第72回ビジネスショウ開催される Windows 3.0，486マシン，DOS/V……

さる5月15日から18日まで，東京・晴海で第72回ビジネスショウが開催された．

日本語Windows 3.0が登場してからはじめての大規模イベントということもあり，各社ともWindows 3.0を前面にだした展示を行っていた．また，高解像度版Windowsを参考出品するところも少なくなかった．さらに今回は，Windowsコンソシアムが出展し，各社のマシンでWindowsアプリケーションのデモを行った．

各ハードメーカーから発表された486マシンも今回の見どころだった．とくに，日本電気がショー直前に発表した486SXを搭載したNESAバスマシンに注目が集っていた．

画像を中心としたマルチメディアは，ネットワークとともに，最近のイベントでは欠かせないものになっているようだ．どのメーカーもマルチメディアとLAN関係の展示を行っていた．

日本IBMは，DOS/Vによる市場開拓に意欲的で，55ノート，高解像度Windows，テラドライブなどでDOS/V環境をアピールしていた．また，パーソナルシステムのソフトウェアを統合するプラットホームとして位置づけたOS/2 2.0を初公開した．

DOS/Vへの対応が注目されるOADG加盟のAXメーカーは，今回はめだった動きを見せず，三菱電機がPS/2互換機のアプリコット，シャープが輸出仕様のPC-6600でDOS/Vを展示した程度だった．

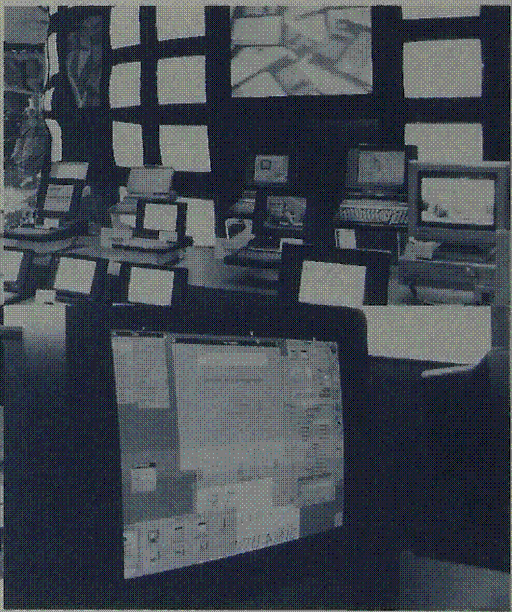
●会場は連日多くの人出で混雑したが，土曜休日が普及したためか最終日はやや減少し，最終的には昨年並とか．

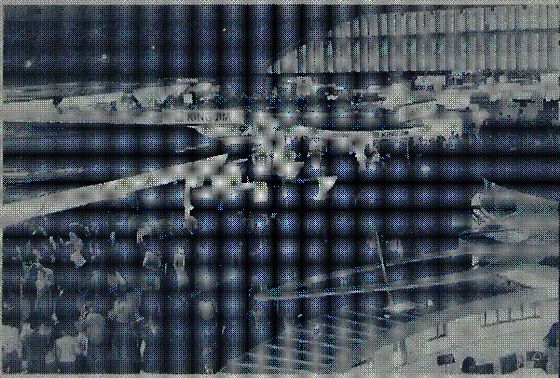
●マシン，ディスプレイ，ソフトウェアでブースの空間をうめつくした東芝．

●日本語環境が整ってきたNeXT．VJEや辞書検索システムなどが出品された．

●DOS，Windows，OS/2，OS/2 PMを統合するプラットホームとされるOS/2 2.0がUS・日本語版ともに初登場．

マシンでうめつくされた東芝ブース▶





▲連日にぎわった会場

日本語環境が整ってきたNeXT▶

アップルコンピュータジャパン，次世代OS「System 7」を発売

アップルコンピュータジャパン（株）（TEL 03-3592-0505）は，Macintoshの新しいOS「System（システム）7」を発売した．

System 7は，既存のアプリケーション，ビット

ワンラインNEWS
東京電子設計はインテルジャパンと共同で，プリンタなどの画像処理機器に内蔵するCPUを検査・修正するためのICEを開発し，販売を開始した．

NEWS & NEW PRODUCTS

マップフォントとの互換性をもち，どのようなフォントサイズでも高品質な画面表示，プリントアウトが可能なTrueTypeフォント技術を採用．ポストスクリプト対応・非対応のプリンタなどでも高品質な出力を得ることができる．Finderは，すべてのFinder機能をダブルクリックのみで開くことができ，ファイル，フォルダのアップルメニューへの追加，フォントやシステムの拡張なども容易に行うことができる．マウスカーソルを移動するだけで「ふきだし」が現れるヘルプシステム「バルーンヘルプ」機能をもつ．ネットワーク上に専用のファイルサーバがなくても，ハードディスクの共有，他機種とのファイルの共有も可能なFile Sharing機能と新たに提供された「エリアス」（フォルダ，ファイルサーバなどにオリジナルのポインタとなるエリアスを設定し，そこからオリジナルファイルを呼び出すことができる）機能を使い，ネットワーク上のどのMacintoshからでも普段と同じように仕事が行える．

また，マルチタスク機能，仮想メモリ（68030，68020搭載マシンのみサポート），8MB以上のRAMを装着しメインメモリとして使用することができる32ビットアドレッシング機能もサポートしている．

アップグレード価格は，HyperCard 2.1 Liteと新機能を説明したスタック「Before You Install」が付属したパーソナルアップグレードキットが16,000円．これに，ネットワークでのインストーラなどを含んだSystem 7 CD-ROMが付属したグループアップグレードキットが55,000円．

日本電気，486SX搭載マシンなどPC-9800シリーズの新製品を発表

日本電気(株)，日本電気ホームエレクトロニクス(株)（NECパソコンインフォメーションセンター：TEL 03-3452-8000）は，国内でははじめて80486SXを搭載し，548,000円からという普及価格を実現した「PC-H98S モデル8」と，現行機（PC-9801NS）にくらべ1.3倍の処理速度を実現した32ビットノート型パソコン「PC-9801NS/E」を発売した．

「PC-H98S モデル8」は，CPUに80486SXを搭載し，現行の32ビット機（PC-9801DA）にくらべて約2倍の処理速度を実現．32ビットバス「NESA」の採用により，16ビットバスの約3.3倍のデータ転送能力をもち，メモリアドレス空間も4GBまで拡大されている．また，データバス幅の自動選択が可能なため，PC-9800シリーズのオプションボードをそのまま使うことができる．1,600万色中16色（オプションで256色）の多色表示，グラフィック専用のVLSI，AGDCとE^2GCの採用により高速なグラフィック描画が可能．

「PC-9801NS/E」（愛称：98NOTE SX/E）は，CPUに80386SXを搭載し，セカンドバッテリパックを装備すれば，FDDモデルで3.4時間，HDDモデルで3.2時間の連続使用が可能となる．オプションのCRTパックを装着すれば，アナログRGBディスプレイを接続することができる．FDDモデルに20MB，40MBのHDDを追加することもできる．

また同時に，CD-ROMユニット，ハードディスク，熱転写プリンタなどの18種のハードウェアと，OS/2関連製品などの23種のソフトウェアも発売した．

■PC-H98Sモデル8/U8

プロセッサ：80486SX（20MHz）／メモリ：1.6MB（最大45.6MBまで拡張可能）／外部記憶装置：5インチFDD×2（8-002）・40MB HDD×1（8-040）・100MB HDD×1（8-100），3.5インチFDD×2（U8-002）・40MB HDD×1（U8-040）・100MB HDD×1（U8-100）／拡張スロット：4（NESA対応，32ビットまたは16ビット）／価格：548,000円(8-002/U8-002)，698,000円(8-040/U8-040)，798,000円（8-100/U8-100）

■PC-9801NS/E

プロセッサ：80386SX（16MHz）／メモリ：640KB（RAMドライブを増設メモリとして使用した場合は1.6MB，最大9.6MBまで拡張可能）／外部記憶装置：3.5インチFDD×1（NS/E）・20MB HDD×1（NS/E20）・40MB HDD×1（NS/E40）／価格：278,000円(NS/E)，378,000円(NS/E20)，418,000円（NS/E40）

NEWS & NEW PRODUCTS

日本IBM，OADG標準機となる低価格のDOS/V用パソコンを発売

日本アイ・ビー・エム（株）（PSサポート・デスクTEL 0120-055302）は，20万円を切った低価格を実現したDOS/V専用デスクトップパソコン「PS/55Zモデル5510Z」とマイクロチャネル採用の32ビットパソコン「同モデル5510T」の2機種を発表した．両機種ともユーザーインターフェースの統一のためキー表示，レイアウトを改善した「OADG」標準キーボードを付属している．

また，同時に80386，80486を搭載したPS/55向けの「OS/2バージョンJ2.0」開発の意向を発表，現行のバージョンJ1.3も36,000円に値下げした．

■PS/55Zモデル5510Z

プロセッサ：80286（12MHz）／メモリ：1MB（最大5MBまで拡張可能）／外部記憶装置：3.5インチFDD×2（Z02/ZJ2），3.5インチFDD×1・40MB HDD×1（Z04/ZJ4）／拡張スロット：3（16ビットフルサイズATバス）／価格：198,000円（Z02），214,000円（ZJ2，DOS/V付属），292,000円（Z04），308,000円（ZJ4，HDDにDOS/Vをインストール済）

■PS/55Zモデル5510T

プロセッサ：80386（20MHz）／メモリ：2MB（最大14MBまで拡張可能）／外部記憶装置：3.5インチFDD×1・40MB HDD×1／価格：538,000円（T04），554,000円（TJ4，HDDにDOS/Vをインストール済）

セガ・エンタープライゼス，2CPUパソコン「テラドライブ」を発売

（株）セガ・エンタープライゼス（テラドライブパソコンサポートダイヤル：TEL 0120-417102）は，日本アイ・ビー・エム（株）との共同開発による，80286と68000の2つのCPUを搭載した2CPUパソコン「テラドライブ」3モデルを発売した．2つの16ビットCPUをもつ「テラドライブ」は，IBM PC用のソフトウェアとメガドライブ用ゲームソフトを同時に動作させることができる．また，バススワップによってCPU同士が結ばれているため，お互いのグラフィックス機能やドライブなどの周辺機器を利用しあうことができる．グラフィックはVGAを搭載し，最大262,144色中256色の表示が可能．OSは日本語DOS/Vを採用し，各種ドライバを搭載したドライバディスクを標準装備している．

■おもな仕様

プロセッサ：80286（10MHz）・68000（7.6/10MHz）・Z80A（音源コントロール用）／メモリ：640KB（MODEL 1），1MB（MODEL 2），2.5MB（MODEL 3）（本体メインボード上のSIMMで最大2.5MBまで拡張可能）／外部記憶装置：3.5インチFDD×1（MODEL 1），3.5インチFDD×2（MODEL 2），3.5インチFDD×1・30MB HDD×1（MODEL 3）／拡張スロット：1（ATバス準拠，ハーフサイズ）／価格：148,000円（MODEL 1），188,000円（MODEL 2），248,000円（MODEL 3，HDDにDOS/Vをインストール済）



日立，VGA搭載のAXマシン「FLORA」シリーズを発売

（株）日立製作所（情報事業本部コンピュータ事業部オープンシステム事業推進部：TEL 03-3763-4838）は，AX-VGA仕様に準拠したAXパソコン「FLORA」シリーズを発売した．「FLORA」シリ

ワンラインNEWS
米アップル社は，米政府が進める周波数の供給不足に伴う「電波の競走入札制」導入に対し，反対する考えを表明した．同社は無線型パソコン通信の普及を進めている．

NEWS & NEW PRODUCTS

ーズは，CPUに80486または80386を搭載し，1120×780ドットの高精細表示と640×480ドットの標準精細表示ができるデュアルスキャンディスプレイ，EISAバスを採用したデスクトップタイプの上位機「FLORA 3020」と，80386SXを搭載したデスクトップタイプとTFT方式のカラー液晶ディスプレイおよび16階調白液晶ディスプレイを搭載したラップトップタイプの下位機「FLORA 3010」の全5機種．また，「3020」には同社の「2020シリーズ」の互換モードが用意されており，「3010」には「B16/32」シリーズの互換モードを標準装備した専用モデルがある．

■FLORA 3020H

プロセッサ：80486（33MHz）／メモリ：4MB（最大20MBまで拡張可能）／外部記憶装置：3.5インチFDD×1・200MB HDD×1／拡張スロット：6（EISA対応フルサイズ，うち1スロットは表示ボードが専有）／価格：2,115,000円

■FLORA 3020F

プロセッサ：80386（33MHz）／メモリ：4MB（最大16MBまで拡張可能）／外部記憶装置：3.5インチFDD×1・80MB HDD×1／拡張スロット：3（EISA対応ハーフサイズ）／価格：1,500,000円

■FLORA 3010D

プロセッサ：80386SX（20／8MHz）／メモリ：2MB（最大16MBまで拡張可能）／外部記憶装置：3.5インチFDD×2（DS20）・50MB HDD×1（DS60）・100MB HDD×1（DS80），3.5インチFDD×1・50MB HDD×1（DS50）／拡張スロット：3（DS50は，専用SCSIスロットあり）／価格418,000円（DS20），608,000円（DS50），618,000円（DS60），758,000円（DS80）

■3010LS

プロセッサ：80386SX（20／8MHz）／メモリ：2MB（最大12MBまで拡張可能）／外部記憶装置：3.5インチFDD×2（LS20），3.5インチFDD×1・40MB HDD×1（LS50）・80MB HDD×1（LS70）／ディスプレイ：16階調白液晶バックライト付／価格：538,000円（LS20），708,000円（LS50），828,000円（LS70）

■3010LST

プロセッサ：80386SX（20／8MHz）／メモリ：2MB（最大12MBまで拡張可能）／外部記憶装置：3.5インチFDD×2（LS25），3.5インチFDD×1・40MB HDD×1（LS55）・80MB HDD×1（LS75）／ディスプレイ：TFT方式カラー液晶バックライト付／価格：798,000円（LS25），968,000円（LS55），1,088,000円（LS75）

サザンパシフィック，dBXL，dBXL/LAN，QuickSilverをバージョンアップ

（株）サザンパシフィック（TEL 045-314-9514）は，米国WORDTECH社のリレーショナルデータベース「dBXL」「dBXL/LAN」，リレーショナルデータベースコンパイラ「QuickSilver」のマルチシステム対応版（NECのPC-9800シリーズ，EPSON PC-286/386，AX，PS/55，J-3100，Panacom M500シリーズ，FM R50の5機種に対応），「バージョン1.4マルチシステムサポート版」を発売した．

「dBXL」は，dBASEIIIの拡張上位互換のリレーショナルデータベースで，「同マルチシステムサポート版」は，実行ファイルとヘルプファイルを圧縮することにより，2HDフロッピーの約半分程度で動作．必要メモリも400KB，EMS使用時には350KBに削減．フロントエンドプロセッサの制御もサポートした．ネットワーク版の「dBXL/LAN」は，上記の機能にくわえ，新たにBROWSE，CHANGEなど50以上のコマンド・関数が追加された．

「QuickSilverバージョン1.4マルチシステムサポート版」は，従来異なった機種で実行するために必要だった，プログラムの再コンパイル，再リンクを不要にし，d-Codeコンパイル，d-Codeリンク，オプティマイズの3つのステップを一本化して開発時間を短縮することができる．また，オーバーレイ・リンカを標準搭載，MS-C 6.0とのリンクインターフェース，フロントエンドプロセッサの制御もサポートした．価格は，69,800円（dBXL），168,000円（dBXL/LAN），198,000円（QuickSilver）

Programmer's Program

高速処理に威力を発揮するグラフィックライブラリ GENIFA

グラフィックを多用するアプリケーションを開発する場合，単純な図形の表示なら，最近のCコンパイラに標準で付属するグラフィックライブラリを使えば処理できる．しかし，図形の移動や拡大などを高速に処理しようとすると，どうしても付属のライブラリではパワー不足を否めない．そこで高速，かつ移植性に優れたグラフィックライブラリが登場することになる．

GENIFAの概要

「GENIFA」は，「Graphics ENvironment InterFAce」の略で，グラフィックソフト開発を強力にサポートするC言語用グラフィックライブラリである．GENIFAは，ユーザーとのマンマシンインターフェースについても工夫されているため，GUIソフトの開発にも威力を発揮する．

GENIFAのパッケージには，5インチまたは3.5インチのフロッピーディスクと，バインダー式のマニュアル1冊が入っている．

マニュアルは，導入方法や概要を簡単に説明した「導入ガイド」，各関数を使用するための説明とアルファベット順の各関数リファレンスを記した「ライブラリ」，付属のユーティリティの操作方法を説明した「ユーティリティ」の3部構成になっている．MS-DOSの基本的なオペレーションを理解し，C言語でのプログラミング経験がある人なら，サンプルプログラムも豊富に掲載されているので，理解するのは難しくない．

マニュアルによると，GENIFAの開発コンセプトは次の3点である．

・高い互換性（移植性のよさ）

GENIFAは，ほとんどがC言語(高速性が必要となる部分はアセンブラ)で記述されているので，さまざまなプラットフォーム(OS，ハードウェア)に移植することができる．GENIFAがプラットフォームに依存した部分を吸収しているので，アプリケーションプログラムはプラットフォームの違いを意識することなく開発できるようになる．

・超高速性

GENIFA内でのハードウェアの直接操作，最適アルゴリズムの採用によって，ハードウェアのもっている最高のスピードをアプリケーションに与えることができる．

・多機能集約性

ライブラリやシェルの目的用途を広げすぎると，メモリの占有率が増加，プログラムが繁雑化，スピードが低下する．GENIFAは機能を集約し，これらの問題を解決している．もちろん，GENIFAにアドオン機能を追加することで，CADなどさまざまな用途向けに機能アップしていくこともできる．

また，GENIFAには開発効率を飛躍的に向上させるユーティリティが用意されている．

GENIFAを使うには，本体であるライブラリファイルGENIFA.LIBをコンパイラの標準ライブラリがインストールされているディレクトリに，ユーティリティをパスの通ったディレクトリに，それぞれコピーすればよい．さらにユーティリティを使うには，マウスドライバを登録する必要がある．NEC版の場合，対応しているドライバはDOSのシステムディスクに付属するMOUSE.SYSだけで，MS-C 6.0に付属するマウスドライバは使えない．この点は近日中にバージョンアップにて対応するということだ．

用意されている関数群

GENIFAには，次のような関数群が用意されている．

・低レベルグラフィック関数

直線・曲線・(楕)円・矩形・角丸矩形・多角形・(楕)円弧の描画，各図形の塗りつぶしパターンや前景色・背景色の塗りつぶし，図形の拡大・縮小，さまざまなフォントパターンでの文字表示，指定域の塗りつぶし，また，仮想座標系と物理座標系をサポートし，いつでも切り換えが可能．とくに，仮想座標系を用いることによって，ハードウェアとは独立した画面イメージを

ワンラインNEWS
日本DECは同社の無停止型コンピュータ「VAXftシリーズ」の上位3モデルを発売した．従来機に比べ2倍近い高速処理が可能となった．

もつことが可能となる．

・ユーザー入力関数

マウス操作やキー入力をイベントとして扱い，発生時間・カーソル位置・キーコード・タイプなど，必要な情報を各イベントとともにバッファに格納し，ユーザープログラムに渡す．

・メニュー，ダイアログボックス関数

メニューバー，プルダウンメニュー，ポップアップメニューなど，8段階まで階層化が可能なメニューシステムや，ユーザーとアプリケーションプログラムとのインターフェースであるダイアログボックスを，必要なデータや処理関数を指定の構造体データで記述するだけで実現できる．さらに，付属のユーティリティを利用することにより，簡単に構造体データソースを自動生成することができる．

ユーティリティ

GUTILS.EXEはメニュー，ダイアログボックス，アイコン，フォント，マウスカーソルなどを編集するためのユーティリティである．

ユーティリティの起動は，MS-DOSのコマンドラインから

```
A> GUTILS
```

と入力し，［RETURN］キーを押す．起動後，ほとんどの操作はマウスを使って行うことができる．

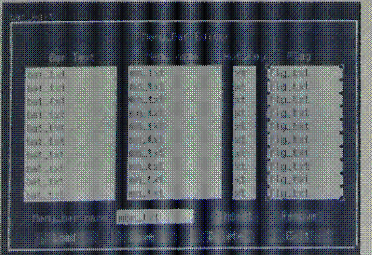
たとえば，ダイアログボックスの設計をする場合，メインメニューから「ダイアログ」をクリックする．画面がクリアされて，空のダイアログボックスが表示される．この状態からマウスの右ボタンを押すと，メニューバーが表示されるので，右ボタンを押したまま「アイテム（I）」を選択する．開いたプルダウンメニューから「追加（A）」を選択すると，さらにサブメニューが開く．ダイアログに追加したいアイテムをサブメニューから選択する．これを繰り返してダイアログを完成させる．

ダイアログボックスのサイズの変更は，マウスをダイアログボックスの右下端に移動し，カーソルの形が両方向の矢印になった状態でドラッグする．

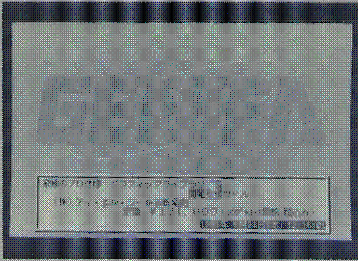
ダイアログボックスが完成したら，メニューバーを表示させ，「ダイアログ（D）」から「終了（E）」を選択し，ダイアログエディタを終了する．

あとはメインメニューから「ソースコードの生成」を選択すれば，ダイアログを開くために必要なデータソースが生成される．

なお，このユーティリティ自身も，GENIFAの各種関数を使って作られているということだ．



ダイアログボックスの設計



デモプログラム

デモプログラム

発売元である（株）アイ・エル・シーの好意により，付録ディスクにGENIFAのデモプログラムを収録した．このプログラムは自己復元形式で圧縮されているので，適当なディスクにコピーし，MS-DOSのコマンドラインから

```
A> GENI9800
```

と入力し，［RETURN］キーを押す（PC-9801用の場合，AX用はGENIAX00）と，いくつかのファイルが復元される．この中にGENIFA98.DOC（PC-9801用の場合，AX用はGENIFAAX.DOC）にデモプログラムの詳しい使用方法が書かれているので，参照して欲しい．

最近各社からグラフィックライブラリがリリースされているが，このGENIFAは高速なグラフィックの描画，文字にも対応した仮想画面，GEMフォーマットに互換性のある文字フォント，移植性の高さなど，ほかのグラフィックライブラリにはない魅力が満載されている．

●GENIFA

販売：（株）アイ・エル・シー／価格131,000円（シングルユース価格，税込み）／動作環境：PC-9801VM2以降（LT，ハイレゾモード，PC-H98は除く），各社AXマシン，EGAまたはVGAを搭載したIBM PC XT/ATおよび互換機／メモリ：640KB以上／動作OS：MS-DOS／対応言語：Microsoft C Ver.5.1，6.0

ワンラインNEWS
富士通は知識獲得のための支援機能を装備した高機能エキスパートシステム構築用ソフトウェア「KwESHEL」を開発，発売した．

オンラインソフト採集帳 7

拡張ディスクコピーツール DCP

「DCP.EXE」は，MS-DOSの標準ツールであるDISKCOPYの機能を強化したもので，EMSメモリやVRAMを利用したコピー，フォーマットしながらのコピー，未使用トラックのスキップ，ファイルを介してのコピー，複数のコピーの作成などの機能をもっている．作者は，本誌の好評連載「プログラム『遊』工房」の筆者でもある谷口典生氏である．

未フォーマットディスクでもOK

DCPは，LHarcで圧縮されているので，復元には吉崎氏が作成した高圧縮書庫管理ユーティリティLHarcまたはLHAが必要になる．LHAまたはLHarcをカレントまたはパスの通ったディレクトリにおき，DCP322.LZHがカレントにあるの確認してMS-DOSのコマンドラインから

```
A> LHA E DCP322
```

と入力する．すると，カレントディレクトリにDCP.EXEとドキュメントファイルDCP.DOCを含むいくつかのファイルが復元される．

使い方はDISKCOPYと同じで，カレントまたはパスの通ったディレクトリにDCP.EXEをおき，MS-DOSのコマンドラインから

```
A> DCP A: B:
```

と入力することにより，Aドライブから Bドライブへディスクのコピーができる．さらに，Bドライブのフロッピーディスクがフォーマットされていない場合は，フォーマットをしてもいいか聞いてくるので，［Y］と入力すればフォーマットしながらディスクをコピーする．また，起動前からフォーマットが必要とわかっている場合には，

```
A> DCP -f A: B:
```

と-fオプションをつければ，同じことになる．フロッピーディスク互換のRAMディスクなどでも問題なく使用できる．

```
A> DCP A: A:
```

と転送元と転送先に同じドライブを指定した場合，起動後メッセージにしたがってフロッピーディスクを入れ換えれば，ノート型パソコンなど1ドライブの機種でもディスクのコピーができる．

DCPは2HD(8セクタおよび15セクタフォーマット)，2DD(8セクタおよび9セクタフォーマット)に対応しており，EMSメモリやVRAMをバッファに使ってコピーすることもできる．

オプションは環境変数に指定できる

DCP.EXEを引数なしで起動すると，オプションの簡単な説明が表示される．デフォルトで指定したいオプションは，環境変数DCPまたは環境設定用ファイルDCP.OPTに記述しておくことができる．たとえば

```
A> SET DCP=febt
```

と環境変数を設定した場合，つねにオプション「febt」が指定されていることになる．

環境設定用ファイルDCP.OPTはカレントディレクトリ，またはDCPが起動されたディレクトリにあればよい．DCP.OPTにはオプションのほかにトラックマップの表示に使われる文字の変更なども指定できる．

対話型のDCPIもある

DCP322.LZHには，DCPのほかにディスクのコピーを対話形式で行うDCPIがいっしょに圧縮されている．MS-DOSのコマンドラインから

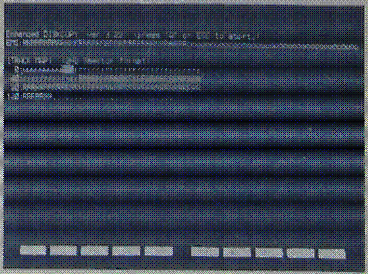
```
A> DCPI
```

と入力するだけである．起動してから，転送元や転送先のドライブやオプションの指定をすることができるので，初心者でも安心して使うことができる．

変わった使い方としては，ドライブの代わりにファイルを指定することで，ディスクイメージをファイルにする機能もある．MS-DOSのコマンドラインから

```
A> DCP B: IMAGE
```

と入力すると，カレントのIMAGE.



DCPというファイルにBドライブのディスクイメージがコピーされる．逆に

A> DCP IMAGE B:

とすればカレントのIMAGE.DCPのディスクイメージをBドライブのフロッピーディスクにコピーすることができる．この機能を使えばPC-9801NS-20など1ドライブのシステムでもフロッピーディスクの入れ換えなしにコピーできるし，また大量のディスクをデュプリケートすることも可能になる．

*

ドキュメントには，今後，デュプリケータとしての機能強化版や，DOS汎用なバージョンの作成も考えていると記されているので大いに期待したい．

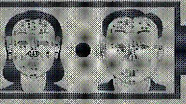
アプリケーションのバックアップなどを行うとき，新品のディスクをフォーマットする手間が省けるのはたいへんうれしい．また，単なるディスクコピーとしても十分に高速である．

■DCP　分類：ディスクコピーツール／ソフト名：DCP／登録名：DCP322.LZH／著作権者：Messiah(谷口典生 pcs31871)／動作環境：PC9801シリーズ／入手場所：ASCII-NET pcs／復元方法：LHA E DCP322

ファイル構成

```
Listing of archive : DCP322.LZH

  Name           Original    Packed  Ratio   Date     Time    Attr Type  CRC
 --------------  --------  --------  ------ -------- -------- ---- ----- ----
  DCP.EXE           24461     14717  60.2% 91-01-21 00:21:40 a--w -lh5- DA6A
  DCP.OPT             340       280  82.4% 91-01-21 00:22:50 a--w -lh5- 70B3
  DCP322.DOC         7523      3548  47.2% 91-01-21 00:30:12 a--w -lh5- CAEF
  DCP1.EXE          29691     17365  58.5% 91-01-21 00:22:34 a--w -lh5- F3E9
 --------------  --------  --------  ------ -------- --------
     4 files        62015     35910  57.9% 91-01-21 00:30:44
```

HEALTHY?  **UNHEALTHY?**

⑦「食品成分表」の効用

「神様，食べても食べても太らないからだをください」「そんなものはありません」……というテレビCMがある．そのとおり，だから○○を飲めば太らないとか，××を食べれば痩せるということはない．

一時流行したコンニャクマンナンも，繊維だけを食べていたのではからだがもたないから，いつのまにかすたれてしまった．酸性・アルカリ性食品という考え方も，もともとが根拠のないものだから，さっぱり聞かなくなった．だいたい，人間のからだは試験管ではないのだから，食物によってPHがコロコロかわってたまるものかと思う．

次々とさまざまな健康法，痩身法，健康食などがでては消える運命にあるのは，「食べても食べても太らないからだ」のように，安易に健康や美しさを獲得したいという願望があるからにほかならない．もちろん，「健康でありたい」「美しくなりたい」という，願いは当然であり，否定されるべきものではない．

データをもっと大切にすべきだと思う．痩身願望をもっている人のうち，『食品成分表』を見たことのある人がどれだけいるだろうか．『○○健康法』などという本を読むより，ずーっと役に立つはずだ．

肉，野菜，魚など素材の成分はもちろん，インスタントラーメンや缶詰などの加工食品，酒類などもデータ化されている．カロリーの項目を見るだけで，たくさん食べると太る食品の見当がつく．なかには，加熱したときのビタミンの歩留まりまででているから，現実に食べる量とビタミンの関係も知ることができる．たとえば，無理をしてレモンを食べるより，加熱したジャガイモをたくさん食べたほうが，楽にビタミンCをとれるというようなことがわかる．

このデータを利用して「健康管理プログラム」なんていうものもできそうだ．でも，データベースを作るのがたいへんだろうな．

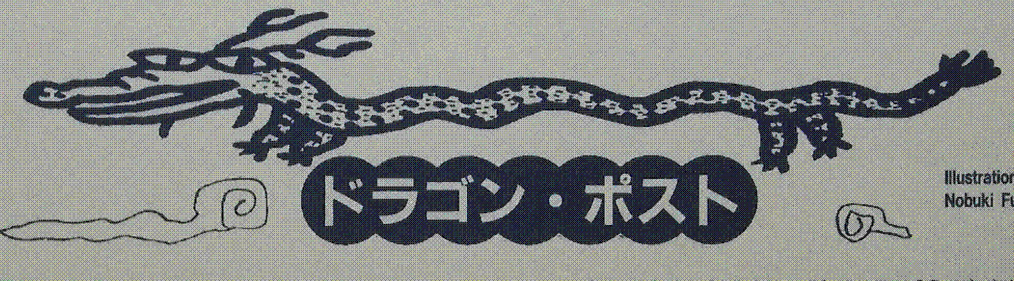


Illustration
Nobuki Fukuhara

この欄は，読者からのお便りを中心に構成しています．本誌に対するご意見・ご感想などをお待ちしております．鬼より恐い読者からの声ということで，「ドラゴン・ポスト」です．本誌についているアンケートはがきにご意見の欄を設けてありますから，そこに書いてください．ペンネームでも歓迎です．ただし，はがきには本名と住所をしっかり書いてくださいね．掲載誌をお届けします．

★Cをはじめてみて……

Turbo Cをはじめたばかりのおじさんです．昔々の８ビットのときにはこんな世の中になるとは思わなかった．Cをやってみて，自分でアプリケーションを作るのは不可能と知りました．ソフトハウスの方々に自分のほしいソフトのアイデアを出すくらいでもできれば，満足できそうです．とかいいつつ，TurboASMものぞきはじめてしまたりして．(横浜久哉)

――自分で作るのは不可能……なんておっしゃっていますが，やっぱり好きなんですね．そのうちにWindowsをインストールして，TP for Windowsをはじめたりして……．

★ああ，あまりのなつかしさに！

NUKEM，ああ，あまりのなつかしさに涙ポロポロものでした．あの当時，まだ私は十代だった．BASICしか知らなかったPC-8001がほしかった（88ではない）．ゲーセンで100円で２回できた「ミサイルコマンド」を当時のカレと向かいあって，ものすごく真剣にトラックボールを転がしていた．あの頃の歌は聞こえない，ヒトの姿もかわったよ，時は流れた．ああ……年くったなあ．(田中はるみ)

――ナツメロじゃない，こういうのはナツゲーというのでしょうか．そのうちに『なつかしのゲーム大全集』なんていう，フロッピーシリーズがでるんじゃないでしょうか？

★OS-9000はどうなったの？

OS-9000の広告が雑誌に出てからだいぶ時間が経つが，一般ユーザー向きの記事が一向に出てこないようである．98もRAからDAになったらしいので，386ベースでマルチタスクOSの"本命"であるOS-9000の記事が読んでみたい．また，OS-9のその後の興味もあります．時々見に行くNIFTYやPC-VANのOS-9フォーラムもわびしい感じですが……．(INT24H)

――表にはあまり知られていませんが，OS-9もまだ活躍しているようですね．それで仕事をしている人にとっては，技術を人に伝えるための参考書などがないことが一番の悩みとか……．

★いま悩んでいます．

パソ通に踏み切るべきか？　で悩んでいます．目的はPDSの利用です．添付のディスクに有用なプログラムがあったので思わず貴誌を買ってしまいました．他誌のディスクサービスでは今一つ「使える」ものがなかったので，不満があったのですが，どうせサービスしてくれるなら，本当に使えるものを添付してくれる貴誌のような内容にしてほしいものです．使い勝手を優先したPDSの配布を今後とも望みます．(石川隆一)

――パソコン通信には，いろいろな面があります．いわゆるPDSもありますし，新しいコミュニティととらえることもできます．とにかく，何かきっかけがあったときに新しい世界に飛び込んでみるというのもおもしろいのでは……．

★MSXを極めたい

大学生でMSXを使用している者の一人です．というのも，別にビジネスソフトを使うでもなし，仕事で使うでもなし，ゲームに飽きてプログラミングでもしてみようかと，MSXの内部を見たらズブズブとはまり込んでしまったわけです．MSXも極めないうちに，ほかのパソコン(ほかのCPU)をいじくる気にもなれず，現状況にあえて甘んじているわけですが……．それから，私は読者の中ではMSXユーザーが最も多いといわれる某バッ活もおもしろいと思っている者の一人です．貴誌を購入したのはいろんなCPUについて知りたいと思っていたからです．(XSM)

――極めるとなれば大変です．

★もう少し初心者にもわかるよう

専門誌ではあるが，付録のゲームなどの使い方（起動およびコピー）をもう少し初心者にもわかるよう解説して欲しいものです．40才を越えてからの手習いは難しいものを感じておりますが，もうしばらくパソコンと付き合ってみようと思います．(Yoshi)

――プログラマのための……なんていっていますが，初心者の方にも読んでいただける何かを，この雑誌がもっ

ているのならうれしいことです．付録の使い方などは，MS-DOSの基本知識に含まれる部分です．でも，それをちゃんと解説するとかなりの分量になってしまいます．そこで，『今夜わかるMS-DOS』（小社刊）などのMS-DOS解説書をおすすめしているのですが……よろしくお願いします．

★世の中，Cばかりだが

世の中C，C，Cばかり．C言語は確かに実行速度は速いが，果たして初心者が使ったところで，あまり使いこなせないのではないだろうか．PascalやBASIC（QuickBASIC），COBOLももう少し見直していいのではないか．ということで，私はPascalのアルゴリズムなどの解説を望みたい（アセンブラの解説ももっともっとやってほしい）．（PUYAN）

――TP for Windowsがでました．MS-BASIC 7.1もでます．アメリカでは，COBOLのWindows開発ツールもでたと聞きます．ということで，C以外の言語もなかなか活発な動きを見せています．

★２度と留年すまい!?

４月も気づいたら後半に入っていた．今日，18日はやたらと風が強くあたたかかった．２限に大ハバにおくれたなと思いつつ本をかかえて歩いていたら，青空が見えるのに雨．きつねのよめいりだー，久しぶり．無事進級できたらできたで，学業がハードになりコンピュータであそべなくてさびしい．辞書引きの毎日から脱出した．とりあえず２度と留年すまいとかたく心にちかったのでありました．今年は単位とるぞ．（四本　恵）

――ということは，コンピュータで遊んでいて一度留年しているのでしょうか？

★１種の受験者は３分の１／

先日，情報処理試験を受けてきました．遅刻者が多くてびっくり．欠席者が多くてびっくり．１種の部屋では全席の1/3も埋まっていなかったんじゃないでしょうか？午前の試験で足切りをするというのが定説ですが，確かにあんなに簡単なのができないなら，午後のを採点しても無駄ですよね．それにしても英語が２問もあるのは解せない!!　選択問題の１つは５×５の魔法陣の穴埋め．これはただのクイズ．次が集合の有名な問題．これもクイズに近い．英語とあわせて４問が一般常識だ．今回の感想は，「頭をひねるような面白い出題がなかった」．初受験なんですがね．（Dannop）

――申込者に対して受験者が少ないとは聞いていましたが，それほどとは．

★パソコンでの開発はきつい？

うーん．パソコンレベルのソフト開発だけがきついと思っていたが，友人やこのページの話によると，ミニコン以上のプログラムもきついらしい．しかし，マシン語とおさらばできるのはとてもよい．パソコンレベルだと，移植性よりスピードを追求し，ついでにプログラマの寿命もちぢめてしまう．もうこんな生活はやだ？……でも抜けられない．（KENJI）

――プログラマの寿命は何歳までなんて，はじめにいっていたのは，大型機の世界だったんじゃないでしょうか？

★マイナーな機種の行方

近頃，久ぶりにマイナーな機種（PC-88VA．知っています？）でBASICインタプリタを使って，プログラミングしています．しかしさすがに遅い．だから一部アセンブラにしようと思い，資料を求めて本屋へ行きました．でも，マイナー機種の悲しさ．資料となる本は絶版．メーカー（NEC）にTELしても「お答えできません．」という返答．どうしたらよいのでしょう？　メーカーはその機種の機能を使えるだけの資料・環境を最低ユーザーに提供する義務があると思います．自分で解析する程Power-Userではない．ということで（どういうことで？）IBMの近頃の姿勢に拍手．（狂夢）

――88VA．16ビットの88として騒がれたやつですね．開発用にMS-DOSもあったと思いましたが，だしたもののサポートはちゃんとやって欲しいものです．

★ソースのないプログラムは味けない

うちにあるソフトといったら，金がないので，雑誌を一所けんめい写経したものとか，パソコン通信でダウンロードしたソフト．言語がよくわからないから，写経したのがすんなり動いたときは絶頂／だけど，まちがいなく写経したはずなのに……動かないのがまま有るんです．次号がきて，ページのかたすみに小さく，ほんとに小さく書いて有るんです．先月×××がバグッてましたうんぬんて，１か月もストレスがたまって，便ぴになって……．最近は，たいていディスクがついてくるからありがたいのですが，しかし……．ソースのないものは味けないですね．どんなおまけがついていても，ソースなしってのは，ほんと味けないです．それに，ソースの横のつまみ（説明）を沢山つけて欲しいものです．わがまま？／（祐子）

――何と／はがき４枚連続のお便り．いままでの愛読者カードをとっておいていただいたのですね．ありがとうございます．

付録ディスクのご案内

このフロッピーディスクに含まれる実行ファイル，ソースリストおよびテキストファイルは「月刊Programmer's Page」1991年７月号に掲載されたものです．

付録のフロッピーディスクは，5"2HD（8セクタ／トラックMS-DOS）でフォーマットされています．3.5"や2DDなどへのコンバートサービスは行っておりません．なお，オンラインサービスSE-NET（03-5276-1297）でも，同じ内容のファイルを用意しています．

フロッピーディスクの内容は次のとおりです．本誌およびディスクに含まれるドキュメントファイルをよく読んでからご利用ください．

●付録ディスクについての説明
ディレクトリ名：ルート
収録ファイル名
README.DOC（テキストファイル）

●特集・最新「デバイスドライバ」研究
①「デバイスドライバ入門」（p. 2）
ディレクトリ名：TOKU1
収録ファイル名

DEVICE.INC	LIST1.ASM
TEST.ASM	LIST2.ASM
TEST1.ASM	LIST3.ASM
TEST2.ASM	LIST4.ASM

②「MS-DOSデバイスドライバの書き方」（p. 19）
ディレクトリ名：TOKU2
収録ファイル名

PRIN2.ASM	DEVDR.INC
RDRV.ASM	PRIN2.SYS
P.BAT	RDRV.SYS
R.BAT	

③「C言語によるデバイスドライバ作成法」（p. 26）
ディレクトリ名：TOKU3
収録ファイル名

MAKETONE.BAT	CHRDRV0.ASM
TONE.SYS	CHRDRV0.H
TESTTONE.EXE	LIST2.ASM
TONE.C	LIST1.ASM
TESTTONE.C	

●プログラム「遊」工房（p. 64）
思考型ゲーム「三子棋（サン・ツー・チ）」
ディレクトリ名：PLAY
収録ファイル名
SANTUTI.EXE　SANTUTI.C

●MS-DOSツール・ラボ（p. 78）
ファイル検索プログラム「T.EXE」
ディレクトリ名：TOOL
収録ファイル名

T.ASM	DOSFUNC.H
TEST.ASM	LIST1.ASM

●98活用アセンブラ技法（p. 90）
ディレクトリ名：98ASM
収録ファイル名

BACK2.ASM	UTILS.INC
BACK3.ASM	BACKDIF.DOC
BACKCOL.ASM	BACKCOL.EXE
CONDX.INC	README.DOC

●実践C塾（p. 102）
ディレクトリ名：JUKU
収録ファイル名

OPTENV.C	TURBOC.H
OPTION.C	STARTP1.C
OPTION.H	LIST4.C
OPTTEST.C	LIST5.C
READCFG.C	LIST12.C
STARTP.C	LIST13.C
STEST.C	

●綴じ込み付録
ディレクトリ名：DATA
収録ファイル名
DOSFUNC.TXT（テキストファイル）

●プログラマーズプログラム（p. 134）
グラフィックライブラリ「GENIFA」体験プログラム
ディレクトリ名：GENIFA
収録ファイル名
GENI9800.EXE　GENIAX00.EXE

●オンラインソフト採集帳（p. 136）
本誌今月号のオンラインソフト採集帳で紹介した多機能ディスクコピーツール「DCP」，このファイルを復元するするのに必要な，高圧縮復元書庫管理ソフト「LHA」（本誌4月号参照）を収録しました．

注意：オンラインソフトは，そのソフトウェアの性格上，該当月号が発売されるまでに新しいバージョンが発表されている可能性があることをご了承ください．また，サポートは基本的にパソコン通信のみで行われておりますので，ご注意ください．

●クイズ・ランダムスクリーン（p. 142）
クイズ解読ツール「PUZLE」

本誌掲載のクイズランダムスクリーンの画像をディスプレイ上に再現し，マウスを使って並び替えることにより，解答を得ることができるユーティリティです．対応機種は日本電気のPC-9801シリーズです．詳しい使い方はPUZLE.DOCを参照してください．

ディレクトリ名：OMAKE
収録ファイル名

CONFIG.SYS	RNDSCREN.G1
AUTOEXEC.BAT	RNDSCREN.B1
PUZLE.EXE	RNDSCREN.E1
PUZLE.DOC	RNDSCREN.RGB
RNDSCREN.R1	

6月号の解答

AJISAI.R1	AJISAI.E1
AJISAI.G1	AJISAI.RGB
AJISAI.B1	

Windows 3.0用 WallPaper（壁紙）
AJISAI.BMP

＊使用上の注意

本誌掲載プログラムおよびデータを利用するには，MS-DOSの基礎知識が必要です．また，このディスクにはMS-DOSのシステムは入っていません．別途MS-DOSが必要です．なお，ソースファイルのコンパイルなどには，別途コンパイラやユーティリティが必要となる場合があります．

バーチャルスペースガイド

P子　某ソフトハウスで企画営業を担当している．26歳．
C平　某メーカーのシステムソフトウェア開発部門に所属．25歳．

C平　テレビの深夜映画を見てて思ったんだけど，80年代前半はなぜかオペラに関係した映画が多かったね．『ディーバ』とか『アマデウス』，そして『フィッツカラルド』．

P子　『ディーバ』は，オペラのアリアも効果的だったけれど，エリック・サティ風のBGMもよかった．

C平　大ヒットはしなかったけれど，業界に影響を与えたんだよね．テレビCMにオペラのアリアが使われるようになったり，ドラマのBGMにサティが流れたり……．ワイドショーの事件再現ドラマってあるでしょ．「人妻A子不倫のはての……」みたいな．ああいうののBGMに聞き覚えのある映画音楽が使われたりして……．

P子　あ，会社遅刻してワイドショー見てるやつ！

C平　ちがうって，たまたま祝日に見ただけだよ．それで，このあいだの深夜映画っていうのが，『フィッツカラルド』だったんだ．

P子　私，それ見てない．確かドイツ映画だったよね．

C平　アマゾンのオペラハウスというのが意表をついている．クラウス・キンスキーがカヌーみたいな小舟でオペラハウスに駆けつけるところからはじまるんだ．あの個性的な顔で……．その相棒で娼家の女将の役をやっているのがクラウディア・カルディナーレ．

P子　それで，アマゾン，オペラハウス……．かなり，味付けが濃いんじゃないですか．

C平　それが，なかなかさわやかな映画なの．とくに，船上でオペラをやるラストシーンがいい．でも，販元に聞いたら廃版なんだって，中古屋さんで探してねって．P子にも見せたかったんだけど．そういえばLDもあったな．

■フィッツカラルド（1982・独）　監督：ウェルナー・ヘルツォーク　出演：クラウス・キンスキー

アマゾンのゴム園で一攫千金を夢見る男フィッツカラルドは，未開の土地を求めてアマゾン川を遡る．オペラを愛する男はレコードをかけながら，インディオしか住まない奥地へ進む．新ルート開拓の切り札は船の山越えだった．船は山を越えたが，激流に流され，夢は消えはてる．

読者プレゼント

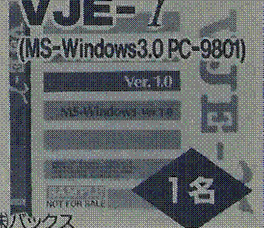
1 GENIFA



2名

提供:㈱アイ・エル・シー

2 VJE-γ (MS-Windows3.0 PC-9801)



1名

提供:㈱バックス

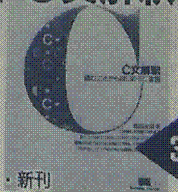
3 バスマウス (PC-9801)



3名

提供:㈱エム・ビー・テクノロジー

4 C文解釈



3名

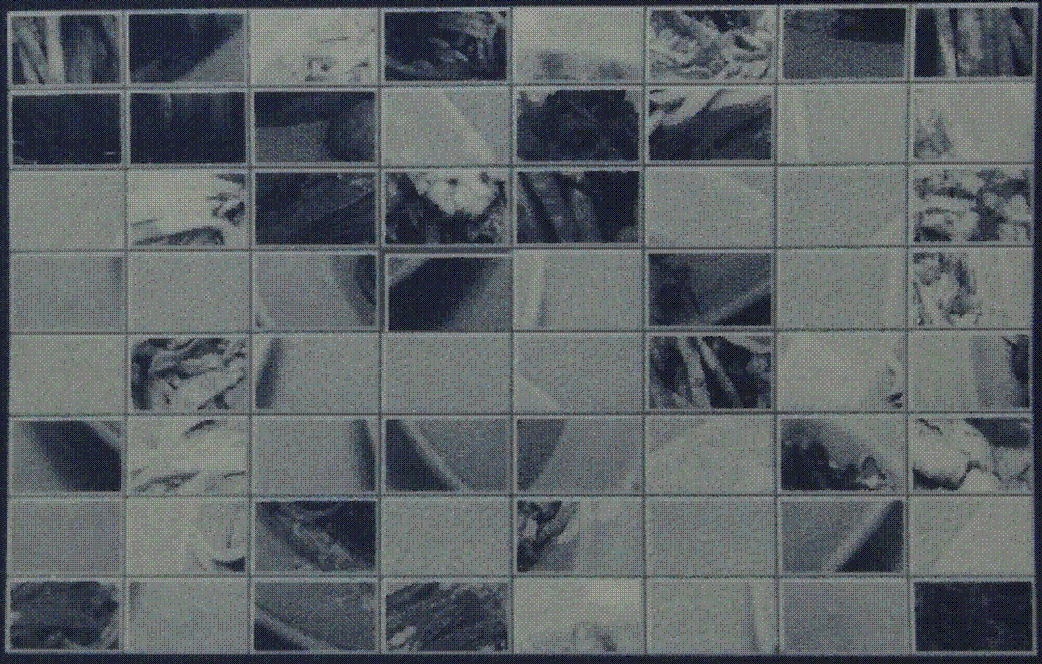
翔泳社・新刊

応募は本誌に添付のアンケートはがきをご利用ください．（1991年7月17日消印有効）．なお，応募者多数の場合は，抽選になります．当選者発表は発送をもって代えさせていただきます．

クイズ・ランダムスクリーン

ある１枚の画像を64分割し，ランダムに並べ替えてあります．
さて，もとの画像は何を写したものでしょうか．

ヒント：夏場の食欲のない時はこれにかぎる．○し○○



正解者の中から30名様に本誌表紙イラスト（日暮修一画）入り月刊プログラマーズ・ページ特製オリジナルマグカップをさしあげます．

応募方法：本誌に添付のアンケートはがきをご利用ください．

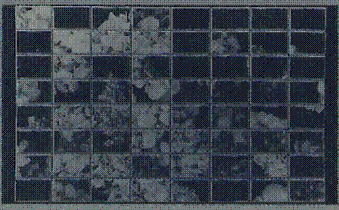
締め切り：1991年７月17日（消印有効）

なお，当選者の発表は発送をもって代えさせていただきます．

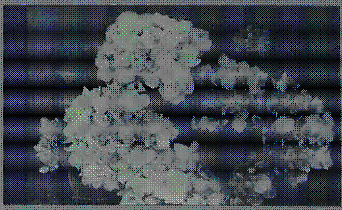
＊画像データは，付録ディスクやBBSでもサービスしています．

6月号の問題と正解

【問題】



【正解：あじさい】



月刊プログラマーズ・ページ オンラインサービス（SE-NET）のお知らせ

付録にフロッピーディスクをつけたため，ディスクによるプログラムサービスは中止いたします．オンラインサービスのBBSは，多数の読者の登録をいただいておりますので，今後も継続いたします．編集部とのコミュニケーションの場としてご利用ください．

●BBSへの接続

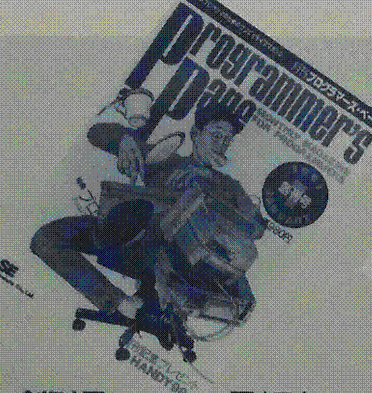
SE-NET　03-5276-1297

①接続するとIDの入力を求めてきますので，GUESTと入力しリターンキーを押してください．
②お知らせを表示後，メインメニューのコマンドプロンプトが表示されます．
③月刊プログラマーズ・ページへのご意見などは「ドラゴンポスト」ボードに書き込んでください．
④ファイルの転送が終了したら，Ｏと入力して回線を切ります．

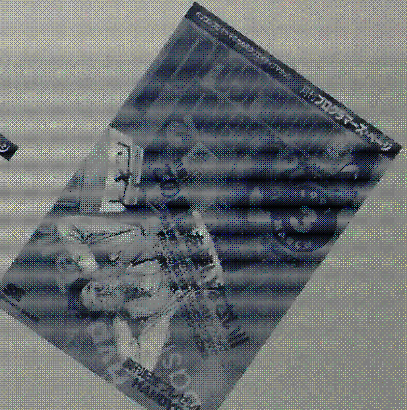
Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

月刊プログラマーズ・ページ バックナンバーのご案内

■定価・各980円







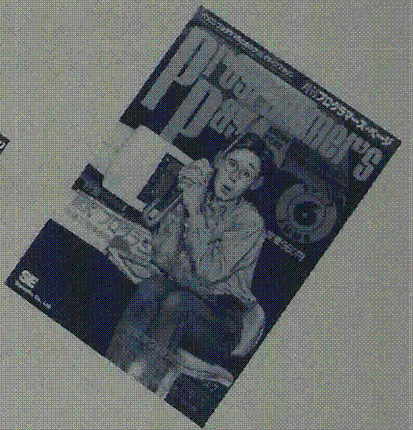
創刊号 品切れ
1991年1月号
特集・プログラマの仕事場
パーソナルコンピュータでソフトウェアを開発する現場の姿をさまざまな角度からレポートし，これまであいまいなイメージしかもたなかったパソコンプログラマ像を描く．

品切れ
1991年2月号
特集・Windowsの時代である!
Windows 3.0の発売で話題の渦中にある新しいOS環境の開発システムを徹底解説．Windowsのサンプルプログラムを関数ごとに図をつけて紹介する．

1991年3月号
特集・この言語を使いなさい!!
数あるプログラミング言語の中から14をピックアップ，それぞれの言語を使い込んだプログラマが，その特性から使い方のポイントまでを解説する．







1991年4月号 5インチFD付き
特集・C読本
パソコンプログラミング言語の標準的存在になったC言語ととことん付き合うための大特集．入門からオブジェクト指向への流れまでを紹介する全5章．

1991年5月号 5インチFD付き
特集・基礎力としてのアセンブラ
プリミティブなプログラミング言語でありながら，それ故に無視することができないアセンブラ．コンピュータを理解し，その能力をひきだすためにもう一度アセンブラをみなおす．

1991年6月号 5インチFD付き
特集・研究「プログラミング統合環境」
今流行りのエディタ，コンパイラ，リンカ，デバッガを統合した開発環境について，その歴史的系譜から最新のシステムまで徹底的に検証する．

「月刊プログラマーズ・ページ」のバックナンバーは，ご注文いただければ，全国どこの書店でもお求めになれます（書店に届くまで1か月ほどかかります）．
小社に直接ご注文なさるときは，何月号かを明記の上，現金書留にて雑誌価格＋送料を下記までお送りください．1月号から5月号の送料は1部260円です（6月号は，1部76円）．2冊以上の場合は，号数・冊数により送料が異なりますので，小社までお問い合わせください．

〒102 東京都千代田区平河町
2-4-14 平河町KSビル4階
（株）翔泳社
マーケティング部PP係宛
TEL 03-3263-0447

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

月刊プログラマーズ・ページ8月号
（7月18日発売）

付録：5″ 2HDフロッピーディスク付き!!

特集
はじめての
オブジェクト指向プログラミング

90年代のプログラミングパラダイムとして注目されている“オブジェクト指向”．このオブジェクト指向について，その歴史的発展の過程から基礎概念，さらにオブジェクト指向分析，オブジェクト指向設計の方法まで網羅的に解説．オブジェクト指向の時代への扉を開く！

月刊プログラマーズ・ページ　1991年7月号
1991年7月1日発行

発行所　株式会社翔泳社
東京都千代田区平河町2-4-14平河町KSビル4F・5F
TEL 03-3263-0447　〒102
発行人　長廻健太郎
編集長　清水　隆
副編集長　鈴木嘉平
編集スタッフ　杉野貴美子＋上田純美礼＋芳山充弘
デザイン　SE（翔泳社デザイン研究所）
広告スタッフ　相澤正夫＋澤本良知＋種村嘉彦
協力　根本裕三郎＋(株)ドモン・マインズ＋佐藤裕美＋小林弘人(図版デザイン)
写植　株式会社東巧社＋株式会社トーヨー・プリント社
印刷　凸版印刷株式会社
定価　980円（本体951円，税29円）

EDITOR'S PAGE

●今年のビジネスショウは，連日たいへんな人手だった．こうしたイベントがあるたびに「今回の目玉は？」という質問を受けるが，いつも答に窮してしまう．ほとんどの製品は，事前に発表されてしまっているし，メーカーの方針と製品の方向性は常にアナウンスされているからだ．あえて見どころといえば，ブースの“見せ方”だろうか．メーカーの力のいれ方や，変化の（あるいは変化しない）兆しを見ることができる．(TA)

●月刊誌の編集部が忙しいのはあたりまえのことだが，それにしても最近やたらに忙しい．ゴールデンウィーク前の混乱を乗り切れば楽になると信じていたのに，まったく楽にならない．自宅で過ごす時間よりも編集部で仕事をしている時間のほうが長いというのは，やはり異常ではあるまいか．「あるまいかって，どんなイカ？」懐かしいギャグだなぁ．というわけで，“脳ミソ”というオブジェクトは“肉体”というオブジェクトの疲労を継承しているのであった．ハハハ……・(嘉平)

●日曜日の夕方，サザエさんがはじまると，「これで休日も終わりだ」と暗くなってしまう『サザエさん症候群』が，世間では流行りだそうです．これって，ついテレビを時計代わりにしてしまう悪習慣の影響もあるのでしょうね．かくいう私も『新世界紀行』を観ながらのアイロンかけが習慣だった時は，あのオープニングテーマを聞いただけで，何となく気が重かったものですが....（ま）

●5月の15日から4日間東京晴海で開かれたビジネスショウでは，多くのパソコンメーカーが「Windows3.0」コーナーを設けて，Windows環境の魅力を訴えていた．だが，そのための環境整備はいま一歩．とくに周辺機器は自社の代表的な製品をサポートするのが精いっぱいのようだ．Windows用のデバイスドライバを自分で書こうにも肝心のWindows DDKが国内で一般に販売されるメドは立っていない．マルチメディアの時代を前に，多くの周辺機器が使えるということはWindowsの大きなメリットにもなるはず．この機に乗じてサードパーティを締めだそうなどという狭い了見を起こすメーカーがないことを祈っています．(kos)

●3月は友人の結婚ラッシュだった．着なれないスーツに身を包んで全国駆け巡り，残ったのは預金通帳のマイナス残高だけだった．あれから2か月が過ぎようとしているが，未だにこの状況から逃れられない．(よ)

愛読者カードご記入のお願い

本誌をお買い上げいただき、誠にありがとうございます。
お手数ですが、今後の誌面づくりの参考にさせていただきたいと存じますので、
各欄にご記入の上、ご返送くださるようお願い申しあげます。

月刊プログラマーズ・ページ READER'S SURVERY CARD 愛読者カード

本誌の号数　　年　　月号

本誌を知った理由
□1.書店店頭　□2.他社出版物（　　　）
□3.小社出版物　□4. DM　□5.口コミ

本誌を購入した書店　　市(区)　　書店

パソコンの所有形態(機種名)
□1.個人所有　（　　　）
□2.職場にある（　　　）
□3.持っていない

よく使用するソフトウェア

よく使用するOS

今月号でおもしろかった記事(いくつでも)

今後取り上げてもらいたい特集のテーマ

貴方が選ぶベスト3
パソコン雑誌 ①　②　③
ソフトウェア ①　②　③
パソコン本体 ①　②　③

本誌についてのご感想、お気づきの点、その他どのようなことでも結構ですので、ご意見をお聞かせください(「ドラゴン・ポスト＝読者の声」欄に掲載させていただく場合もあります。ペンネームをご希望の方はあわせてご記入ください)。

月刊プログラマーズ・ページ 広告資料請求カード

8	会社(学校)名		9	所属(学部)		10	従業員数	

※下記の該当する番号をご記入ください。

11	業種		12	職種		13	パソコン利用状況	

業種
①製造業　②金融・保険・不動産業　③商社・卸・小売業
④建設業　⑤通信・運輸　⑥システム・ソフト
⑦その他サービス業　⑧公務員　⑨その他

職種
①コンピュータ(ハード)技術者　②コンピュータ(ソフト)技術者　③エレクトロニクス技術者
④プログラマ　⑤その他の技術者　⑥専門職
⑦事務・営業職　⑧経営管理職　⑨商工サービス
⑩教職　⑪学生　⑫その他

パソコン利用状況　①会社で利用　②個人で利用　③会社でも個人でも利用　④利用していない

●希望される広告資料　　年　　月号

広告資料請求No. / 希望項目												
A 購入希望	A	A	A	A	A	A	A	A	A	A	A	A
B 説明希望	B	B	B	B	B	B	B	B	B	B	B	B
C 資料希望	C	C	C	C	C	C	C	C	C	C	C	C

定期購読のご案内

綴じ込みの振替用紙を使って、本誌の定期購読をお申し込みになれます。
1年間12冊=11,760円(本体11,417円/税343円)
郵送料サービス

郵便はがき

102-00

料金受取人払
麹町局承認
3908

差出有効期間
平成4年11月
30日まで

切手不要

東京都千代田区平河町2-4-14
平河町KSビル
株式会社 翔泳社
マーケティング部 行

月刊プログラマーズ・ページ
READER'S SURVEY CARD　愛読者カード

氏名	性別 □1男 □2女	生年月日 19　年　月　日
住所　〒		電話 (　　)　－

職業			ユーザー区分
□1ソフト技術者	□2ハード技術者	□3技術系会社員	□1プロ
□4事務系会社員	□5営業系会社員	□6教師	□2マニア
□7一般公務員	□8経営管理職	□9理系学生	□3一般ユーザー
□10文系学生	□11その他(　　)		

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

郵便はがき

102-00

料金受取人払
麹町局承認
3910

差出有効期間
平成4年11月
30日まで

切手不要

東京都千代田区平河町2-4-14
平河町KSビル
株式会社 翔泳社
月刊プログラマーズ・ページ
広告資料請求係行

＊当社で記入

1	SE	2	受付No.

月刊プログラマーズ・ページ広告資料請求カード

3	会員番号	

＊すでに会員番号をお持ちの方はご記入ください。

4	フリガナ 氏名	姓　　名	5	生年月日	19　年　月　日

6	自宅住所	〒　電話　(　　)	資料送付希望先の番号に○印をつけてください。
7	会社住所	〒　電話　(　　)	

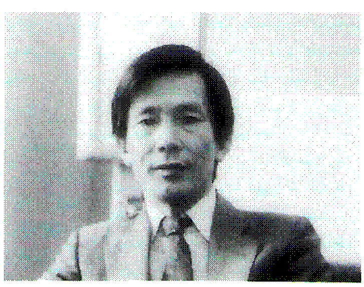
数理技研

術計算や基本ソフトを中心
フトウェアの研究開発に取り
か実績は宇宙開発から、UN
N、HDTV、スーパーコン
ーキング、そして企業の勤務
まで、実に幅広い。しかも、す
でのプロジェクト展開であ
くでは経験できないような知
60名すべてがSEであり、高
社の採用方針、教育内容と
くの矢掛弘司総務部長を訪ね

者は新卒3名、中途1名という内容です。新卒のうちコンピュータ関係の専門学校卒が4名、工学系の大学院卒が1名です。

私どもの業務は高度であるため、今まで条件を『大卒以上』と示してきました。今年専門学校卒を採用したのは、これまで11年間培ってきた教育システムが確立されてきたからです。もちろん試験と面接はありますけど、ある程度の能力と教養のある人材であれば育て上げる自信がついたといえるでし

れだけ情報が拡充は早く、コンピュータの最先端分野に携わっている当社としましては、新たな情報や技術に対して常に研究する必要がありますからね。



の講座は、週に1回以上午前中2の学習を必須とするもので、UNやMACHのソースコード解析にる講座などを設け、コンピュータエンス中心に先輩社員から理論にる徹底した指導を受けます。」

配属はどのように決められるので？

新規・中途にかかわらず、入社と同時に仮の受け入れ場所として特定のプロジェクトに配属になり、1～3ヵ月間はほとんど勉強に終始してもらいます。そこでマネージャーやプロジェクトリーダーからどういうシステムでどのように仕事をしているか、当社の開発環境を早く知ってもらうわけです。また一つのプロジェクトは個人の能力を生かすために小編成のチームを採っていますので、マンツーマンのかたちでの教育が行われます。

払込票

通常払込料金加入者負担

口座番号 東京 4－154493

加入者名 株式会社 翔泳社

金額 11760円

払込人住所氏名 ※

備考

受付局日附印

切り取らないで郵便局にお出しください。

記載事項を訂正した場合は、その箇所に訂正印を押してください。

払込通知票

通常払込料金加入者負担

口座番号 東京 4－154493

加入者名 株式会社 翔泳社

金額 11760円

料金　特殊　円　払込み

備考

受付局日附印

払込人住所氏名 ※（郵便番号　　）

TEL（　　）

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。（郵政省）

各票の※印欄は、払込人において記載してください。

定期購読のご案内

■定期購読は、1年間12冊と限定させていただきます。

■購読料のお支払は、この振替用紙をご利用ください。銀行振込、現金書留によるご送金は、ご遠慮ください。

■振替用紙に、ご住所、お名前をご記入のうえ、郵便局の貯金窓口にてお支払ください。

■小社より領収書の発行はいたしませんので、郵便局にて「郵便振替払込金受領証」をお受取ください。

■お申し込みいただいた方には、弊社より直接発送いたしますので、到着が発売日より遅れることがございますが、ご了承ください。

月刊プログラマーズ・ページ　定期購読直接申し込み書

この欄は、加入者あての通信にお使いください。

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

年　　月号より

年間購読料 11,760 円(税・送料込み)

■ご希望の購読開始月号をご記入ください。

■送付先（住所・氏名は正確にご記入ください）

住所	□□□-□□	フリガナ	
		氏名	
		TEL	

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。　（郵　政　省）

郵便はがき

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社

月刊プログラマーズ・ページ
広告資料請求係行

2	受付No.	

マーズ・ページ広告資料請求カード

＊すでに会員番号をお持ちの方はご記入ください。

名	5	生年月日	19　年　月　日

綴じ込みの

□10文系学生　□11その他(　　　　)　ユーザー

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

6	自宅住所	〒　電話（　　）	○資料送付希望先の番号に印をつけてください。
7	会社住所	〒　電話（　　）	

人材募集
広告企画

ちょっと
気になる
この会社

株式会社 数理技研

1979年に設立以来、科学技術計算や基本ソフトを中心に、最先端分野で独創的なソフトウェアの研究開発に取り組んできた数理技研。その活動実績は宇宙開発から、UNIXなど新世代OS、ISDN、HDTV、スーパーコンピュータ、異機種間ネットワーキング、そして企業の勤務管理システムや会計システムまで、実に幅広い。しかも、すべて現代テクノロジー最前線でのプロジェクト展開であり、同社には他のソフトハウスでは経験できないような知的緊張がみなぎっている。社員60名すべてがSEであり、高度な研究者を目指すという同社の採用方針、教育内容とは？　新宿にある同社オフィスの矢挂弘司総務部長を訪ねてみた。

1年を3年もの実績にする充実の教育

――現在、貴社ではどのように人材を採用しているのですか？

「今までは中途採用の方が多かったのですが、5年程前から新規採用の公募を行っています。しかしネームバリューのあるところに流れてしまって、なかなかいい人材が確保できないのが現状です。」

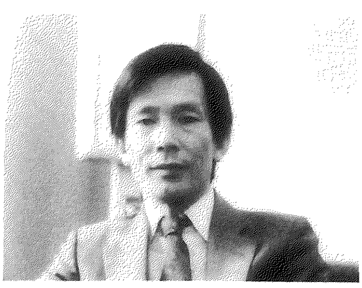
――春の採用実績は、新規と中途ではどのような割合ですか？

「今春は新卒5名、中途1名という内容です。新卒のうちコンピュータ関係の専門学校卒が4名、工学系の大学院卒が1名です。

私どもの業務は高度であるため、今まで条件を『大卒以上』と示してきました。今年専門学校卒を採用したのは、これまで11年間培ってきた教育システムが確立されてきたからです。もちろん試験と面接はありますけど、ある程度の能力と教養のある人材であれば育て上げる自信がついたといえるでしょう。実際に今まで何回か他社との比較で調査したことがあるのですが、当社に1年勤めれば、他社での3年位の実績を積ませることができますよ。」

――貴社が自信を誇るその教育システムとはどのような内容なのでしょう。

「それは『数理アカデミア』といいます。2年前に、それまで空いた時間に行っていた勉強会を勤務時間内に持ち込もうということから始めました。これだけ情報が活発な中で、コンピュータの最先端分野に携わっている当社としましては、新たな情報や技術に対して常に研究する必要がありますからね。



この講座は、週に1回以上午前中2時間の学習を必須とするもので、UNIXやMACHのソースコード解析に関する講座などを設け、コンピュータサイエンス中心に先輩社員から理論に関する徹底した指導を受けます。」

――配属はどのように決められるのですか？

「新規・中途にかかわらず、入社と同時に仮の受け入れ場所として特定のプロジェクトに配属になり、1～3ヵ月間はほとんど勉強に終始してもらいます。そこでマネージャーやプロジェクトリーダーからどういうシステムでどのように仕事をしているか、当社の開発環境を早く知ってもらうわけです。また一つのプロジェクトは個人の能力を生かすために小編成のチームを採っていますので、マンツーマンのかたちでの教育が行われます。

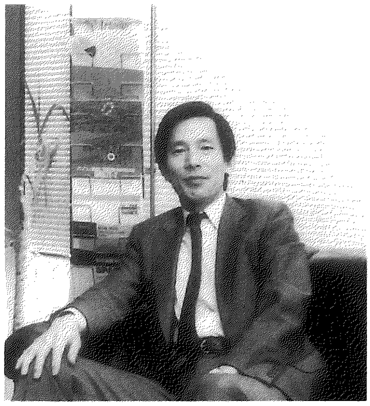
3言語をマスターする必要がある

――入社と同時に配属が決まると、希望とは違うなどの不満は出ませんか？

「それはありません。というのも、私どもでは面接の際に本人の実績や希望をリサーチし、包み隠さずに話し合い互いのコンセンサスを得た上で配属を決めていますから。そのため、定着率が非常に高いことは当社の誇れる一つでしょう。また最初に決めた配属先は固定ではなく、2～3ヵ月後にプロジェクトリーダーが技術の習熟度や分野の適性をチェックするなど審査して、社員それぞれが確実に技術力を向上できるような体制をつくっています。」

――貴社の開発に必要な言語はどういうものがありますか？

「私どもでは〝三ヵ国語〟として、FORTRAN、C言語、COBOL(もしくはAI言語）をマスターすることを必須としています。ユーザが私どもに希望しているのは、こういうシステムならこういう言語を使ってというように、一から提案できる総合力あるSEなんです。ですから数理アカデミアをはじめ研修期間が終わっても1～3年はOJTを導入して、これらの言語をひと通りマスターしてもらいます。一番難しいのは中途採用の場合にそれまで培った設計手法をなかなか払拭できない人がいることです。うちの諸先輩が築いたプログラム手法にいかに早く合わせられるか、その意味では若くて柔軟性があり、しかも全体的にまとまりある仕事をこなせる人材が望まれます。」



エキスパートへ、到達主義を採用

――社員の待遇面はどうなっているのでしょう？

「男女の格差は全くありません。給与体系は二つありまして、一つは到達主義、一つは実績主義です。多くの社員は前者で、後者はマネージャークラスです。到達主義には4段階あり、初級SE、中級SE、上級SE、スーパーSEへと順に上がっていきます。最終目標を10点満点とし、そこに達すれば実績主義となります。そして技術力に応じた柔軟な待遇を行います。また最低でも1年で1ポイントは上がるよう年齢の問題も考慮しています。とにかくいいプログラム、いいロジックが書ける人材、それに管理能力、マネジメント能力が備われば、うちではエキスパートとして何より優遇されます。」

――貴社ではフレックスタイム制も導入されているようですが？

「例えば大きなプロジェクトが終わった翌日は、あまり早く出社してもしかたがありませんからね。いつもフレッシュな気持ちと元気な体で仕事に取り組んでほしい。そのために時間管理はすべて個人まかせです。」

――矢掛部長から見て社内の雰囲気はどんな感じですか？

「当社では社長とか部長とかの肩書きは一切使わず、さん付けやニックネームで呼び合っています。堅苦しい上下関係は全くありません。また、一つのプロジェクトが終わったときの打ち上げには社員全員に声がかかるなど、和気藹々としています。

そのほか、当社では、アメリカの動向を調べるためにショウなどに出席する費用や、関連書籍の購入にしてもすべて会社負担にしています。PCやワークステーションなどの充実したハード機器と併せて、社員への投資は全く惜しみませんよ。

研究のための受け皿はある会社です。ですから、少々生意気でもいいです。大学の数学や物理の教養程度があり、自己主張できるだけの個性豊かな人材に期待したいと思います。うちでの業務が務まれば、どんなソフト開発も怖いものなしになりますよ。」

●

豊富な理論学習の機会に象徴されるように、数理技研は大量採用の促成栽培を否定、新人を一人前のエンジニアとして徹底的に鍛え上げる少数精鋭主義を貫いている。技術者が可能な限り研究開発に専念できるようなシステムは、知的好奇心に富み、新しく困難なシステム開発に意欲ある人には魅力的だろう。単なるサラリーマンに満足しない、開発への情熱あふれる研究者肌の人に最適の環境ではないだろうか。

川辺義宏クンは、数理技研をどう考えているか。

㈱数理技研は、今回大々的に、強力なスタッフを多数募集します。これは、そのための人材募集シリーズ広告・第1弾です。興味をもたれた方のご応募をお待ちしています。



第1話「数理アカデミア」でトホホの巻

僕が数理技研にきたのは、ほんの1年前のこと。
それまでは、普通のソフトウエアハウスで、普通のプログラムを組む、普通のプログラマだった。
特に不満があったわけじゃないけど、なんかもっと
「高度な仕事がしたい！」っていう欲求が膨らんできたある日、数理技研のことを耳にした。
「スゴそうな会社だな」というのが、第一印象だった。
入社後すぐに、「数理アカデミア」という、これもまたスゴそうな名前の講座を毎日受けることになった。
正直いって、ショックだった。それまで僕が、
多少のプライドをもって蓄積してきたソフトウエアの知識は、みごとに打ち砕かれた。
——ＵＮＩＸやＭＡＣＨのソースコード解析、オブジェクト指向、ＣＧレイトレーシングなど、
コンピュータ・サイエンスを中心とした理論学習の数々。
「ムム、この会社は、目標としているモノが違う」
単なるプログラマの育成を行おうとしているのではない。さまざまな技術分野で、
自らが第一線として開発を行う、研究者・研究企業を目指しているのだ。
その意欲は数理技研の社内のいたるところで感じられ、
大きなプレッシャーとして僕の上にのしかかってきたのであった。
（次号へ続く）

DATA

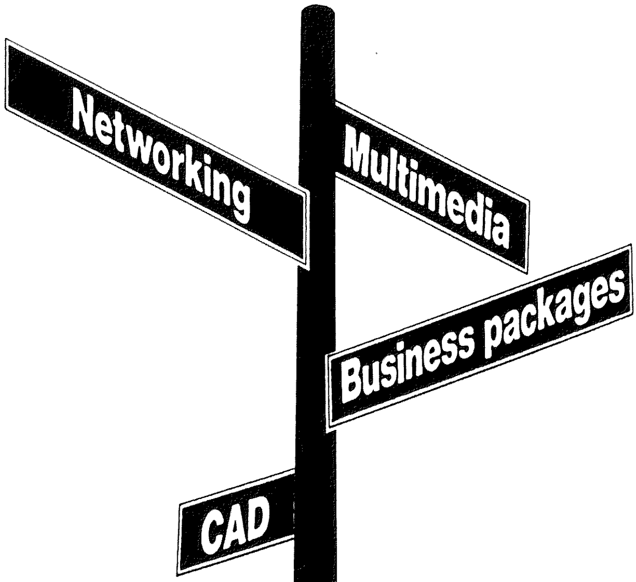
■募集要項
職種●システムエンジニア
資格●大卒以上もしくは大卒程度の学力を有する28才位までの男女。
※やる気・好奇心があり、技術を吸収する力がある方。
給与●経験・年齢を考慮の上、当社規定により優遇します。（'91年新大卒基本給18万３千円）
待遇●昇給年１回（'91年度実績平均8.0%）、賞与年２回（'90年度実績平均7.2ヵ月）、各種社会保険完備、交通費全額支給、諸手当（子供・役職）、制度（住宅購入資金融資・退職金）、施設（独身寮・リゾートマンション）
休日休暇●完全週休２日制（土日）、祝日、年末年始、夏期休暇
勤務地●東京および山梨（'92年４月開設）
勤務時間●フレックスタイム制
教育制度●数理アカデミアと称する体系的な理論学習の講座制度あり。技術セミナーへの参加や論文発表の機会多数。
応募●履歴書をご郵送下さい。追って詳細をご連絡します。
交通●JR新宿駅南口下車徒歩２分

■会社概要
設立●1979年３月
資本金●4800万円
売上高●11億円（'90年８月期）、13億円（'91年８月期見込み）
社員数●60名（男45名、女15名）
事業内容●①UNIXはじめOS・通信系基本ソフトの開発
②大規模・分散処理システムの開発と同インテグレーション
③上記①、②の研究開発
④宇宙開発・核融合・数値解析など、科学技術計算の先端分野での研究開発
特徴●企業というより技術研究機関
促成栽培を否定した技術者教育
開発環境の完備
タイムカードなし、服装自由など自主性尊重
事業所●本社：東京都新宿
研究所：山梨県竜王
※'92年４月、山梨県竜王町のソフトウエアパーク内に山梨研究開発センターを開設

主要取引先●宇宙科学研究所、日本原子力研究所、NTT、NTTデータ通信、地球科学総合研究所、日本IBM、富士通、富士通研究所、日本電気、本田技研、日産自動車、IHI、住友電工、日立製作所、東芝、ソニー、アンリツ、日本信号、日本無線、ファナック、ヤマハ、NKKなど
技術提携先●UNIXインターナショナル、中国科学院軟件研究所、MENTAT Inc.（米）

株式会社 数理技研
〒160 東京都新宿区新宿4-1-9 新宿ユースビル
TEL 03-3356-2781㈹





高水準の技術開発には パーフェクト精神が根幹です。

ワンランク上の提案型ソフトウェアハウス＝エルテックスは、ユーザーや上司の求めている以上の開発を追求していく、我がままプロ集団の組織。常に高レベルの開発を行っていくためには、たえず外部の開発にアンテナを張りめぐらしておかなくてはなりません。その一環として、毎年数名、ラスベガスのコムデックスショー等、1週間程度の視察旅行に参加します。パーフェクトなものを求め続けていく技術者たちに、私たちは、たえず評価を与えていきます。

技術者随時募集

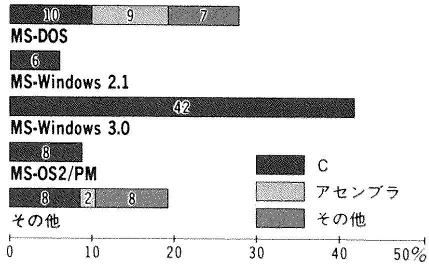
会社概要 ELTEX

設　立	昭和60年12月14日
資本金	1,000万円
事業内容	パソコンを中心としたシステムの提案・開発・販売　①CD-ROM・LD など新しいメディアに対応したシステムの開発 ②ネットワークの端末システムの開発 ③特定業種向けCADの開発・販売 ④基本ソフトの開発
特　徴	大手クライアント向けのシステム開発が多く、ネットワーク関連システムやマルチメディアなど幅広く対応している。また、ＣＡＤを中心とした自社プロダクツにも積極的で、その比率も増加している。基本ソフトの分野では、ハードウェア専門のベンチャー企業とジョイントし、ＯＥＭ製品のＢＩＯＳ・ユーティリティなどを企画から手掛けている。
取引先	三菱電機、三菱自動車、日本航空　他
関連会社	株式会社ソーテック、学校法人福井情報ビジネス総合学院

募集要項 ELTEX

職　種	①システムエンジニア　②プログラマー
資　格	専門卒以上　20～36歳の方　※Ｃ言語・アセンブラ言語経験者大歓迎　※未経験者でも好奇心・向学心の旺盛な方は歓迎します。
給　与	17万円（専門卒初任給）～46万円（36歳課長、諸手当含む） 24歳例：23万5000円（情報処理１種手当・住宅手当含む） 30歳例：31万6000円（中級SE、情報処理１種・住宅手当含む）
手　当	時間外手当、役職手当（5000円～６万円）、住宅手当（１万5000円）、資格手当（２種：5000～特種：２万円）、通勤手当
賞　与	賞与年２回（平成１年度実績年間５ヵ月）、決算賞与年１回（平成１年度実績１ヵ月）
勤務地	本社（横浜駅近く） ◇派遣・出向はありません。
勤務時間	フレックスタイム制（コアタイム10:00～16:00）試用期間中は標準時間9:00～17:45
休日休暇	完全週休２日制（土・日）、祝日、年３回連休（GW・夏期・年末年始）、有給・特別・休暇
福利厚生	各種社会保険完備、退職金制度、財形貯蓄、社員旅行、海外研修、定期健康診断、社員寮、スポーツクラブ法人会員
応募方法	電話連絡の上、履歴書(写真貼付)、職務経歴書を持参し来社ください。郵送も受け付けます。
連絡先	〒220 横浜市西区南幸二丁目20番５号　東伸24ビル５階　℡(045) 322-9667　総務課

最近のOS・言語別開発動向



採用HOT・LINE
TEL.(045) 322-9667

株式会社エルテックス

BORLAND

キミが、ターボ・エンジン。

誕生から2年。まだまだ若いボーランドジャパン。
入社後、すぐに全力疾走してもらうことになるかもしれない。
言語界のエポック・メイキング・ソフト、あの「TURBOシリーズ」を生んだ米ボーランド社の
日本での“顔”になるわけだから、プレッシャーも大きい。
でも、「それがどーした!」のパワーがあれば、今までとは桁違いの自信と充実感が、きっと返ってくる。
ボーランドジャパンを加速するターボ・エンジンたち。
鼻息も荒く、やって来い!



募集要項

職　種　①ソフトウェア移植スタッフ
※米ボーランド インターナショナル社のプロダクトを国産PCに移植
②テクニカル・サポート・エンジニア
③テクニカル・ライター
④営業

資　格　①アセンブラ、C経験者(英語力があれば、なお可)
②③④経験者優遇
※学歴不問、年齢18～30歳位までの方

給　与　当社給与規定により優遇。経験者、固定給20万～40万円

待　遇　昇給年1回、賞与年2回、交通費全額支給、各種社会保険完備、退職金制度

勤務時間　①10:00～18:00
②③④9:00～17:00

休日休暇　完全週休2日制、祝日、年末年始、夏期、有給、特別休暇

応　募　電話連絡の上、履歴書(写真貼付)をご持参ください。郵送も可。

交　通　渋谷駅よりバス(都01、新橋駅行)5分南青山7丁目下車
地下鉄表参道駅より徒歩12分

採用関係連絡先　〒107 東京都港区南青山7-8-1
小田急表青山ビル9階
TEL.03-3486-1400(大代表)

会社概要
●設　　立:平成元年4月
●資 本 金:3000万円
●事業内容:ソフトウェアの開発・販売

株式会社ボーランドジャパン

職　種　初級SE・プログラマー
資　格　高卒以上　20歳～28歳位迄
※大学生歓迎　※機種・言語は問いません。
給　与　月給　16万円以上
※経験・前職を考慮して、相談の上決定します。
待　遇　昇給年１回、賞与年２回(昨年度実績６ヵ月)、交通費全額支給、各種社会保険完備、手当／家族・住宅(２万円)・残業・役職、退職金制度、社内旅行（家族も同伴で行けます。昨年プーケット５日間)
勤務地　本社(中央区)
勤務時間　９:00～17:30
※自己申告により前後１時間ずれてもOK！
※標準労働時間37.5h　※残業月平均20時間
休日休暇　週休２日制　土、日、祝(月１回土曜出社)、夏期・年末年始・GW(各々９連休)・有給（休暇以外に初年度６日使用　次年度より１日ずつ加算
※有給消化率90%以上)・慶弔・社内旅行休暇
応　募　電話連絡の上、履歴書持参（写真貼付）ください。郵送も可。※入社日相談に応じます。※応募の問い合わせ歓迎します。
交　通　地下鉄日比谷線・都営浅草線人形町駅徒歩３分
地下鉄都営新宿線浜町駅徒歩４分
地下鉄半蔵門線水天宮前駅徒歩５分
事業内容　メインフレームからパーソナル・コンピュータまでのソフトウェア開発及びサポート。データ通信システムの設計及び開発。コンピュータ関連機器の輸出入・販売・導入コンサルティング及び技術サポート。経営合理化に関するコルサルテーション。
機　種　IBM 3090・AS/400、
NEC N-3100・PC/9801、他多数
言　語　COBOL、PL/1、RPG、C言語、アセンブラ



遅刻という言葉はありません。

プロとして………その1

私たちはプロ集団です。プロとしての自覚は自己管理能力にあります。１週間の開発スケジュールを作り、時間管理もしていきます。出社時間も自己管理していくから、遅刻という言葉はありません。だけど、社会人としてのモラルは最低限守っていきたいですね。

プロとして………その2

私たちはプロ集団です。プロとしての役割は、よりよいものを開発していくことにあります。そのためには個性を生かし、ユーザーに対して自己主張（高度な提案）をしていかなくてはなりません。仕事も真剣に、遊びも真剣に………これがプロとしての条件です。だけど、真剣になりすぎて感情的になってはいけませんよ。

株式会社 ユニコット UNICOT　TEL 03(3249)0920
〒103 東京都中央区日本橋人形町2-29-9 ISビル5F
学生応募も大歓迎

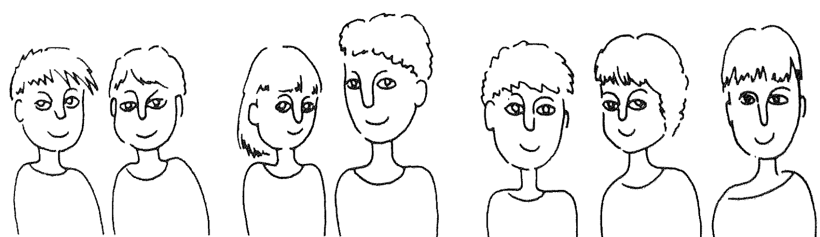
ロジック・アーツ株式会社

事業内容　基本ソフトの開発からパソコン・マイコンの応用システムの企画・開発・販売、電子回路の設計及び製造まで幅広く手掛けています。
職　種　SE・プログラマー
※派遣は一切ありません。
資　格　高卒以上　20歳～30歳位
給　与　月給　22万円以上　経験・年齢考慮の上、当社規定により決定。
勤務時間　９:30～18:00
休日休暇　完全週休２日制(土・日)、祝日、夏期・年末年始・有給・特別・慶弔休暇
待　遇　昇給年１回、賞与年２回、交通費全額支給、各種社会保険完備
応　募　電話連絡の上、履歴書持参ください。
面接日・入社日応相談
交　通　有楽町線・東武東上線志木駅徒歩7分
言語・OS　CP/M、MS-DOS、UNIX、アセンブリ言語(Z80、64180、8086、68000)、C言語(hitech-c、ms-c、TURBO-c、etc.)
※HP-IB、RS-232c で市販計測器または特注製作品とパソコンによる自動計測システムには数多くの経験があります。近年ではワークステーションによる画像処理も手掛けています。)

〒352　埼玉県新座市東北2-13-25 城取ビル2F
TEL.0484(74)2225 担当/鈴木

これから何をやろうか

「高品質な製品を効率よく開発する」
設立当時(４年前)に掲げた我が社のテーマ。
そして夢は一歩近づいた。
協力しあう技術者集団と広義に渡る開発環境が
「夢実現」への近道となった。
後はどれだけ実績を積むか、
どれだけ組織の「和」を広げられるかにかかっている。
そのためには君たちの参加が必要となってきた。
君たちが我が社に訪れたとき
君たちの夢を聞きたい。
そして一緒に
「これから何をやろうか」考えていきたい。



技術者随時募集

SE 翔泳社が贈る プログラミングブックス

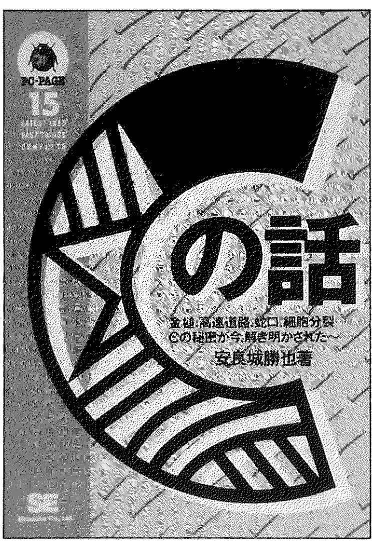


アセンブラプログラミング撤底解説宣言
アセンブラプログラマー必携の書登場!

マクロアセンブラ プログラマーズテキスト

吉川敏則 著■定価2,600円(税込)

PC-9801シリーズの代表的なCPUであるインテル社のi8086系列(i80286, i80386)および標準オペレーティングシステムのMS-DOSを活用するためのマクロアセンブラ(MASM)を使ったプログラミング書。最低限度必要なハードとソフトの知識から、データ演算の仕組みを踏まえた上で、アセンブラプログラミングの基本を丁寧に解説します。また後半では、実践的なアセンブラテクニックをMS-DOSファンクション・リクエストの活用法から学び、Code Viewによるデバッグ方法で締めくくります。



C言語入門書異説

Cの話

安良城勝也 著■定価1,500円(税込)

赤ん坊が言葉を学ぶときは、「はじめに文法ありき」ではない。言葉のシャワー、情景のシャワーの中でセンスが磨かれ、突如として天啓が訪れる。プログラミング言語を学ぶにも、まずセンスの醸成が必要だ。本書はC言語を学ぶ前に読む本。ソフトウェアとは何か、Cのプログラムとはどんなものか…軽いタッチで一気に読める、頭の準備体操。

SE 株式会社 翔泳社
東京都千代田区平河町2-4-14 平河町KSビル 〒102
PHONE 03-3263-0447 FAX 03-3263-4526

〈資料請求番号004〉

MS-DOS Version 3.3 ファンクションリクエスト一覧

ZF＝ゼロフラグ，CF＝キャリーフラグを示し，1でセット，0でリセットを示す．

（PC-9801版）

00H	Terminate Program（プログラムの終了）→4CH
01H	Read Keyboard And Echo（キーボード入力とエコー） （呼） AH＝01H （返） AL＝入力された文字
02H	Display Character（文字のスクリーン出力） （呼） AH＝02H DL＝スクリーン出力する文字 （返） なし
03H	Auxiliary Input（補助入力） （呼） AH＝03H （返） AL＝補助装置から入力された文字
04H	Auxiliary Output（補助出力） （呼） AH＝04H DL＝補助装置に出力する文字 （返） なし
05H	Print Character（文字のプリンタ出力） （呼） AH＝05H DL＝プリンタに出力する文字 （返） なし
06H	Direct Console I/O（直接コンソールI/O【入力】） （呼） AH＝06H DL＝FFH （返） ZF＝1 AL＝キャラクタ 0 AL＝00H（入力されていない）
	Direct Console I/O（直接コンソールI/O【出力】） （呼） AH＝06H DL＝出力する文字 （返） なし
07H	Direct Console Input（直接コンソール入力） （呼） AH＝07H （返） AL＝標準入力（キーボード）から入力された文字
08H	Read Keyboard（キーボード入力） （呼） AH＝08H （返） AL＝標準入力（キーボード）から入力された文字
09H	Display String（文字列のスクリーン出力） （呼） AH＝09H DS:DX＝'$'で終り，スクリーンに出力する文字列のポインタ （返） なし
0AH	Buffered Keyboard Input（バッファドキーボード入力） （呼） AH＝0AH DS:DX＝入力バッファのポインタ （返） なし
0BH	Check Keyboard Status（キーボードステータスの検査） （呼） AH＝0BH （返） AL＝FFH（文字が入っている）/00H（文字が入っていない）
0CH	Flush Buffer, Read Keyboard（バッファを空にして，キーボード入力） （呼） AH＝0CH AL＝01H,06H,07H,08H,0AH（対応するMS-DOSファンクションリクエストの実行）/ほかの値（これ以上の処理は行われない） （返） AL＝00H（バッファを空にした）
0DH	Reset Disk（リセットディスク） （呼） AH＝0DH （返） なし
0EH	Select Disk（ディスクの選択） （呼） AH＝0EH DL＝ドライブ番号（00H＝A:,01H＝B:,…） （返） AL＝論理ドライブの数
0FH	Open File（ファイルのオープン）→3DH
10H	Close File（ファイルのクローズ）→3EH
11H	Search For First Entry（最初のエントリを検索）→4EH
12H	Search For Next Entry（次のエントリを検索）→4FH
13H	Delete File（ファイルの削除）→41H
14H	Sequential Read（シーケンシャルリード）→3FH
15H	Sequential Write（シーケンシャルライト）→40H
16H	Create File（ファイルの作成）→3CH,5AH,5BH
17H	Rename File（ファイル名の変更）→56H
19H	Get Current Disk（カレントディスクを取得する） （呼） AH＝19H （返） AL＝カレントドライブ番号（00H＝A:,01H＝B:,…）
1AH	Set Disk Transfer Address（ディスク転送アドレスの設定） （呼） AH＝1AH DS:DX＝ディスク転送アドレスのポインタ （返） なし
1BH	Get Default Drive Data（デフォルトドライブのデータを取得する） （呼） AH＝1BH （返） AL＝1クラスタ当たりのセクタ数 CX＝1セクタ当たりのバイト数 DX＝1ドライブ当たりのクラスタ数 DS:BX＝FAT-IDのポインタ
1CH	Get Drive Data（ドライブのデータを取得する） （呼） AH＝1CH DL＝ドライブ番号（00H＝カレント,01H＝A:,…） （返） AL＝FFH（ドライブ番号の指定が無効）/1クラスタ当たりのセクタ数 CX＝1セクタ当たりのバイト数 DX＝1ドライブ当たりのクラスタ数 DS:BX＝FAT-IDのポインタ
21H	Random Read（ランダムリード）→3FH
22H	Random Write（ランダムライト）→40H
23H	Get File Size（ファイルの大きさを取得する）→42H
24H	Set Relative Record（相対レコードの設定）→42H
25H	Set Interrupt Vector（割り込みベクタの設定） （呼） AH＝25H AL＝割り込みタイプ番号 DS:DX＝割り込み処理ルーチンのポインタ （返） なし
26H	Create New PSP（新しいPSPを作成する）→4B00H
27H	Random Block Read（ランダムブロックリード）→3FH
28H	Random Block Write（ランダムブロックライト）→40H
29H	Parse File Name（ファイル名の解析） （呼） AH＝29H AL＝解析の制御 DS:SI＝解析するストリング ES:DI＝オープンされていないFCB （返） AL＝00H（ワイルドカードが使用されていない）/01H（ワイルドカードが使用されている）/FFH（ドライブ文字が無効） DS:SI＝解析したストリングの直後のアドレス ES:DI＝オープンされていないFCB
2AH	Get Date（日付を取得する） （呼） AH＝2AH （返） CX＝年(1980～2099) DH＝月(1～12) DL＝日(1～31) AL＝曜日(00H＝日，01H＝月，…06H＝土)
2BH	Set Date（日付を設定する） （呼） AH＝2BH CX＝年(1980～2099) DH＝月(1～12) DL＝日(1～31) （返） AL＝00H(有効な日付)/FFH(無効な日付)
2CH	Get Time（時刻を取得する） （呼） AH＝2CH （返） CH＝時(0～23) CL＝分(0～59) DH＝秒(0～59)
2DH	Set Time（時刻を設定する） （呼） AH＝2DH CH＝時(0～23) CL＝分(0～59) DH＝秒(0～59) DL＝00H （返） AL＝00H（有効な時刻)/FFH（無効な時刻）
2EH	Set/Reset Verify Flag（ベリファイフラグの設定／解除） （呼） AH＝2EH AL＝00H（解除)/01H（設定） DL＝00H （返） なし
2FH	Get Disk Transfer Address（ディスクの転送アドレスを取得する） （呼） AH＝2FH （返） ES:BX＝ディスク転送アドレスのポインタ
30H	Get MS-DOS Version Number（MS-DOSのバージョン番号を取得する） （呼） AH＝30H （返） AL＝バージョン番号の整数部 AH＝バージョン番号の小数部 BH＝OEMのシリアル番号 BL:CX＝24ビットのユーザーシリアル番号（OEMによって異なる）
31H	Keep Process（キーププロセス） （呼） AH＝31H AL＝抜け出しコード DX＝パラグラフ（16バイト単位）でのメモリサイズ （返） なし
33H	CTRL-C Check（CTRL-C検査の状態の取得） （呼） AH＝33H AL＝00H （返） AL＝FFH（ファンクションが無効） DL＝00H（CTRL-C検査が設定されていない)/01H（CTRL-C検査が設定されている）
	CTRL-C Check（CTRL-C検査の設定） （呼） AH＝33H AL＝01H DL＝00H（CTRL-C検査の解除)/01H（CTRL-C検査の設定） （返） AL＝FFH（ファンクションが無効）
35H	Get Interrupt Vector（割り込みベクタを取得する） （呼） AH＝35H AL＝割り込み番号 （返） ES:BX＝割り込みルーチンのポインタ
36H	Get Disk Free Space（ディスクのフリースペースを取得する） （呼） AH＝36H DL＝ドライブ番号（00H＝カレント,01H＝A:,02H＝B:,…） （返） BX＝使用可能なクラスタ数 DX＝1ドライブ当たりのクラスタ数 CX＝1セクタ当たりのバイト数 AX＝FFFFH（ドライブ番号が無効)/1クラスタ当たりのセクタ数
38H	Get Country Data（国別情報を取得する） （呼） AH＝38H AL＝情報を取得する国のカントリーコード（00H＝現在の国，01H＝USA，51H＝日本） DS:DX＝情報に対する32バイトのバッファのポインタ （返） CF＝0（エラーなし） 1 AX＝02H（無効なカントリーコードが指定された）
	Set Country Data（国別情報を設定する） （呼） AH＝38H DX＝FFFFH AL＝FFH/カントリーコード BX＝カントリコード（AL＝FFHの場合） （返） CF＝0（エラーなし） 1 AX＝02H（無効なカントリーコードが指定された）
39H	Create Directory（ディレクトリの作成） （呼） AH＝39H DS:DX＝パス名を示すASCIZ文字列のポインタ （返） CF＝0（エラーなし） 1 AX＝03H（無効なパス)/05H（アクセス拒否）
3AH	Remove Directory（ディレクトリの削除） （呼） AH＝3AH DS:DX＝パス名を示すASCIZ文字列のポインタ （返） CF＝0（エラーなし） 1 AX＝03H（無効なパス)/05H（アクセスの否定)/10H（カレントディレクトリ）
3BH	Change Current Directory（カレントディレクトリの変更） （呼） AH＝3BH DS:DX＝パス名を示すASCIZ文字列のポインタ （返） CF＝0（エラーなし） 1 AX＝03H（無効なパス）
3CH	Create Handle（ハンドルの作成） （呼） AH＝3CH DS:DX＝パス名を示すASCIZ文字列のポインタ CX＝属性 （返） CF＝0 AX＝ファイルハンドル 1 AX＝03H（無効なパス)/04H（オープンされているファイルが多すぎる)/05H（アクセスの否定）
3DH	Open Handle（ハンドルのオープン） （呼） AH＝3DH AL＝ファイルアクセスコントロール DS:DX＝パス名を示すASCIZ文字列のポインタ （返） CF＝0 AX＝ファイルハンドル 1 AX＝01H（無効なファンクションコード)/02H（ファイルが存在しない)/03H(無効なパス)/04H(オープンしているファイルが多すぎる)/05H（アクセスの否定)/0CH（無効なアクセス）
3EH	Close Handle（ハンドルのクローズ） （呼） AH＝3EH BX＝ファイルハンドル （返） CF＝0（エラーなし） 1 AX＝06H（無効なハンドル）
3FH	Read Handle（リードハンドル） （呼） AH＝3FH DS:DX＝バッファのポインタ CX＝読み込むバイト数 BX＝ファイルハンドル （返） CF＝0 AX＝読み込まれたバイト数 1 AX＝05H（アクセスできない)/06H（ハンドルが無効）
40H	Write Handle（ライトハンドル） （呼） AH＝40H DS:DX＝バッファのポインタ CX＝書き込むバイト数 BX＝ファイルハンドル （返） CF＝0 AX＝書き込まれたバイト数 1 AX＝05H（アクセスできない)/06H（ハンドルが無効）
41H	Delete Directory Entry（ディレクトリエントリの削除） （呼） AH＝41H DS:DX＝パス名を示すASCIZ文字列のポインタ （返） CF＝0（エラーなし） 1 AX＝02H（無効なファイル)/05H（アクセスの否定）
42H	Move File Pointer（ファイルポインタの移動） （呼） AH＝42H CX:DX＝オフセット AL＝00H（ファイルの先頭からオフセットを加えた位置に移動)/01H（現在の位置からオフセットを加えた位置に移動)/02H(ファイルの終端にオフセットを加えた位置に移動) BX＝ファイルハンドル （返） CF＝0 DX:AX＝新規のポインタロケーション 1 AX＝01H（無効なファンクション)/06H（無効なハンドル）
43H	Get File Attributes（アトリビュートを取得する） （呼） AH＝43H DS:DX＝パス名を示すASCIZ文字列のポインタ AL＝00H （返） CF＝0 CX＝属性 1 AX＝01H（無効なファンクション)/02H（無効なファイル名)/03H（無効なパス)/05H（アクセスの否定）
	Set File Attributes（アトリビュートを設定する） （呼） AH＝43H DS:DX＝パス名を示すASCIZ文字列のポインタ AL＝01H CX＝属性 （返） CF＝0（エラーなし） 1 AX＝01H（無効なファンクション)/02H（無効なファイル名)/03H（無効なパス)/05H（アクセスの否定）

4400H

Get IOCTL Data (IOCTLデータを取得する)
(呼) AX=4400H BX=ハンドル
(返) CF=0 DX=デバイスデータ
| AX=01H (無効なファンクション)/06H (無効なハンドル)

4401H

Set IOCTL Data (IOCTLデータを設定する)
(呼) AX=4401H BX=ハンドル DX=デバイスデータ
(返) CF=0 DX=デバイスデータ
| AX=01H (無効なファンクション)/06H (無効なハンドル)

4402H

Receive IOCTL Character (IOCTLキャラクタを受け取る)
(呼) AX=4402H BX=ハンドル CX=コントロールデータのバイト数
DS:DX=バッファのポインタ
(返) CF=0 AX=転送されたバイト数
| AX=01H (無効なファンクション)/06H (無効なハンドル)

4403H

Send IOCTL Character (IOCTLキャラクタを送る)
(呼) AX=4403H BX=ハンドル CX=コントロールデータのバイト数
DS:DX=バッファのポインタ
(返) CF=0 AX=転送されたバイト数
| AX=01H (無効なファンクション)/06H (無効なハンドル)

4404H

Receive IOCTL Block (IOCTLブロックを受け取る)
(呼) AX=4404H BL=ドライブ番号 (00H=カレント, 01H=A:, 02H=B:…)
CX=コントロールデータのバイト数 DS:DX=バッファのポインタ
(返) CF=0 AX=転送されたバイト数
| AX=01H (無効なファンクション)/05H (無効なドライブ番号)

4405H

Send IOCTL Block (IOCTLブロックを送る)
(呼) AX=4405H BL=ドライブ番号 (00H=カレント, 01H=A:, 02H=B:…)
CX=コントロールデータのバイト数 DS:DX=バッファのポインタ
(返) CF=0 AX=転送されたバイト数
| AX=01H (無効なファンクション)/05H (無効なドライブ番号)

4406H

Get Input IOCTL Status (入力ステータスのチェック)
(呼) AX=4406H BX=ハンドル
(返) CF=0 AL=00H (レディ状態ではない)/FFH (レディ)
| AX=01H (無効なファンクション)/05H (アクセスの否定)
/06H (無効なハンドル)/0DH (無効なデータ)

4407H

Get Output IOCTL Status (出力ステータスのチェック)
(呼) AX=4407H BX=ハンドル
(返) CF=0 AL=00H (レディ状態ではない)/FFH (レディ)
| AX=01H (無効なファンクション)/05H (アクセスの否定)
/06H (無効なハンドル)/0DH (無効なデータ)

4408H

IOCTL Is Changeable (IOCTLの交換性)
(呼) AX=4408H BL=ドライブ番号 (00H=カレント, 01H=A:, 02H=B:…)
(返) CF=0 AX=00H (交換可能)/01H (交換不可能)
| AX=01H (無効なファンクション)/0FH (無効なドライブ番号)

4409H

IOCTL Is Redirected Block (IOCTLリディレクトブロック)
(呼) AX=4409H BL=ドライブ番号 (00H=カレント, 01H=A:, 02H=B:…)
(返) CF=0 DX=デバイスアトリビュートワード
| AX=01H (無効なファンクション)/0FH (無効なドライブ番号)

440AH

IOCTL Is Redirected Handle (IOCTLリディレクトハンドル)
(呼) AX=440AH BX=ハンドル
(返) CF=0 DX=IOCTLビットフィールド
| AX=01H (無効なファンクションコード)/06H (無効なハンドル)

440BH

IOCTL Retry (IOCTLリトライ)
(呼) AX=440BH DX=リトライの回数 CX=待ち時間
(返) CF=0 (エラーなし)
| AX=01H (無効なファンクションコード)

440CH

Generic IOCTL (for handles) (一般IOCTL (ハンドル用))
(呼) AX=440CH BX=ハンドル CH=05H(カテゴリコード) CL=ファンクション
(マイナー) コード DS:DX=データバッファへのポインタ
(返) CF=0 (エラーなし)
| AX=01H (無効なファンクションコード)

440DH

Generic IOCTL (for block devices) (一般IOCTL (ブロックデバイス用))
(呼) AX=440DH BL=デバイス番号 (00H=カレント, 01H=A:, 02H=B:…)
CH=08H (カテゴリコード) CL=ファンクション (マイナー) コード
DS:DX=パラメータブロック-Iへのポインタ
(返) CF=0 (エラーなし)
| AX=01H (無効なファンクションコード)/02H (無効なドライブ)

440EH

Get Logical Drive Map (論理ドライブマップの取得)
(呼) AX=440EH BX=ドライブ番号 (00H=カレント, 01H=A:, 02H=B:…)
(返) CF=0 AL=00H (物理的にマップされている)
| AX=01H (無効なファンクションコード)/0FH (無効なドライブ番号)

440FH

Set Logical Drive Map (論理ドライブマップの設定)
(呼) AX=440FH BX=ドライブ番号 (00H=カレント, 01H=A:, 02H=B:…)
(返) CF=0 AL=00H (物理的にマップされた)
| AX=01H (無効なファンクションコード)/0FH (無効なドライブ番号)

45H

Duplicate File Handle (ファイルハンドルの二重化)
(呼) AH=45H BX=ファイルハンドル
(返) CF=0 AX=新規のファイルハンドル
| AX=04H(オープンされているファイルが多すぎる)/06H(無効なハンドル)

46H

Force Duplicate File Handle (ファイルハンドルの強制二重化)
(呼) AH=46H BX=既存のファイルハンドル CX=新規のファイルハンドル
(返) CF=0 (エラーなし)
| AX=04H(オープンされているファイルが多すぎる)/06H(無効なハンドル)

47H

Get Current Directory (カレントディレクトリを取得する)
(呼) AH=47H DS:SI=64バイトのメモリ領域に対するポインタ
DL=ドライブ番号 (00H=カレント, 01H=A:, 02H=B:…)
(返) CF=0 (エラーなし)
| AX=0FH (無効なドライブ)

48H

Allocate Memory (メモリの割り当て)
(呼) AH=48H BX=割り当てるメモリサイズ (パラグラフ)
(返) CF=0 AX=割り当てられたメモリのセグメントアドレス (パラグラフ)
| AX=07H(メモリ中のデータの破壊)/08H(十分なサイズのメモリがない) BX=割り当て可能なメモリサイズ

49H

Free Allocated Memory (割り当てられたメモリの解放)
(呼) AH=49H ES=解放するメモリのセグメントアドレス
(返) CF=0 (エラーなし)
| AX=07H (メモリ中のデータの破壊)/09H (無効なブロック)

4AH

Set Block (割り当てられたメモリブロックの変更)
(呼) AH=4AH ES=変更するメモリのセグメントアドレス
BX=変更するメモリのサイズ (パラグラフ)
(返) CF=0 (エラーなし)
| AX=07H(メモリ中のデータの破壊)/08H(十分なサイズのメモリがない)/09H (無効なブロック)

4B00H

Load and Execute Program (プログラムのロードと実行)
(呼) AX=4B00H DS:DX=パス名を示すASCIZ文字列のポインタ
ES:BX=パラメータブロックのポインタ
(返) CF=0 (エラーなし)
| AX=01H (無効なファンクション)/02H (ファイルが存在しない)/04H (オープンされているファイルが多すぎる)/05H (アクセスの否定)/08H (十分な大きさのメモリがない)/0AH (不正な環境)/0BH (不正なフォーマット)

4B03H

Load Overlay (オーバーレイのロード)
(呼) AX=4B03H DS:DX=パス名を示すASCIZ文字列のポインタ
ES:BX=パラメータブロックのポインタ
(返) CF=0 (エラーなし)
| AX=01H (無効なファンクション)/02H (ファイルが存在しない)/04H (オープンされているファイルが多すぎる)/05H (アクセスの否定)/0AH (不正な環境)

4CH

End Process (プロセスの終了)
(呼) AH=4CH AL=リターンコード
(返) なし

4DH

Get Return Code Child Proccess (子プロセスからリターンコードを取得する)
(呼) AH=4DH
(返) AX=抜け出しコード

4EH

Find First File (最初に一致するファイル名の検索)
(呼) AH=4EH DS:DX=パス名を示すASCIZ文字列のポインタ CX=属性
(返) CF=0 (エラーなし)
| AX=02H (パス名が無効)/12H (一致するファイルがない)

4FH

Find Next File (つぎに一致するファイル名の検索)
(呼) AH=4FH
(返) CF=0 (エラーなし)
| AX=12H (一致するファイルがない)

54H

Get Verify State (ベリファイの状態を取得する)
(呼) AH=54H
(返) AL=00H (ベリファイオフ)/01H (ベリファイオン)

56H

Change Directory Entry (ディレクトリエントリの変更)
(呼) AH=56H DS:DX=既存ファイルのパス名を示すASCIZ文字列のポインタ
ES:DI=新規のパス名を示すASCIZ文字列のポインタ
(返) CF=0 (エラーなし)
| AX=02H (ファイルが存在しない)/05H (アクセスの拒否)/11H (装置の不一致)

57H

Get Date/Time of File (ファイルの日付/時刻を取得する)
(呼) AH=57H AL=00H BX=ファイルハンドル
(返) CF=0 CX=時刻 DX=日付
| AX=01H (無効なファンクション)/06H (無効なハンドル)

Set Date/Time of File (ファイルの日付/時刻を設定する)
(呼) AH=57H AL=01H BX=ファイルハンドル CX=時刻 DX=日付
(返) CF=0 (エラーなし)
| AX=01H (無効なファンクション)/06H (無効なハンドル)

58H

Get Allocation Strategy (アロケーションストラテジを取得する)
(呼) AH=58H AL=00H
(返) CF=0 AX=00H(下位)/01H(最小)/02H(上位)
| AX=01H(無効なファンクションコード)

Set Allocation Strategy (アロケーションストラテジを設定する)
(呼) AH=58H AL=01H BX=00H(下位)/01H(最小)/02H(上位)
(返) CF=0 (エラーなし)
| AX=01H(無効なファンクションコード)

59H

Get Extended Error (拡張されたエラーコードを取得する)
(呼) AH=59H BX=00H
(返) AX=拡張されたエラーコード BH=エラークラス BL=可能な対処
CH=ローカス
(CL,DX,SI,DI,BP,DS,ESの各レジスタの内容は破壊される)

5AH

Create Temporary File (一時ファイルの作成)
(呼) AH=5AH CX=アトリビュート DS:DX=パス名を示すASCIZ文字列のポインタ
(返) CF=0 AX=ファイルハンドル
| AX=03H (パスが存在しない)/05H (アクセスできない)

5BH

Create New File (新しいファイルの作成)
(呼) AH=5BH CX=アトリビュート DS:DX=パス名を示すASCIZ文字列のポインタ
(返) CF=0 AX=ファイルハンドル
| AX=03H (パスが存在しない)/04H (オープンするファイルが多すぎる)/05H (アクセスできない)/50H (ファイルがすでに存在する)

5C00H

Lock (ファイルアクセスロック)
(呼) AX=5C00H BX=ファイルハンドル
CX:DX=ファイルのロックされた領域のポインタ SI:DI=領域のサイズ
(返) CF=0 (エラーなし)
| AX=01H (無効なファンクションコード)/06H (無効なハンドル)/21H (ロックの破壊)

5C01H

Unlock (ファイルアクセスロック解除)
(呼) AX=5C01H BX=ハンドル CX:DX=ロックを解除する領域のポインタ
SI:DI=領域のサイズ
(返) CF=0 (エラーなし)
| AX=01H (無効なファンクションコード)/06H (無効なハンドル)/21H (ロックの破壊)

5E00H

Get Machine Name (マシン名を取得する)
(呼) AX=5E00H DS:DX=バッファのポインタ
(返) CF=0 CX=ローカルコンピュータの番号
| AX=01H (無効なファンクションコード)

5E02H

Printer Setup (プリンタセットアップ)
(呼) AX=5E02H BX=割り当てリストのインデックス
CX=セットアップ文字列の長さ
DS:SI=セットアップ文字列のポインタ
(返) CF=0 (エラーなし)
| AX=01H (無効なファンクションコード)

5F02H

Get Assign List Entry (割り当てリストのエントリを取得する)
(呼) AX=5F02H BX=割り当てリストのインデックス DS:SI=ローカル名のためのバッファのポインタ ES:DI=リモート名のためのバッファのポインタ
(返) CF=0 BL=03H (プリンタ)/04H (ドライブ) CX=ユーザー変数域
| AX=01H(無効なファンクションコード)/12H(これ以上のファイルはない)

5F03H

Make Assign List Entry (割り当てリストのエントリの作成)
(呼) AX=5F03H BL=03H (プリンタ)/04H (ドライブ) CX=ユーザー変数域
DS:SI=ソースデバイス名のポインタ
ES:DI=ディスティネーションデバイス名のポインタ
(返) CF=0 (エラーなし)
| AX=01H (無効なファンクションコード)/03H (パスが見つからない)
/05H (アクセスできない)/08H (メモリ不足)

5F04H

Cancel Assign List Entry (割り当てリストのエントリの解除)
(呼) AX=5F04H DS:SI=ソースデバイス名のポインタ
(返) CF=0 (エラーなし)
| AX=01H (無効なファンクションコード)/0FH (サーバ上のリディレクトの中止)

62H

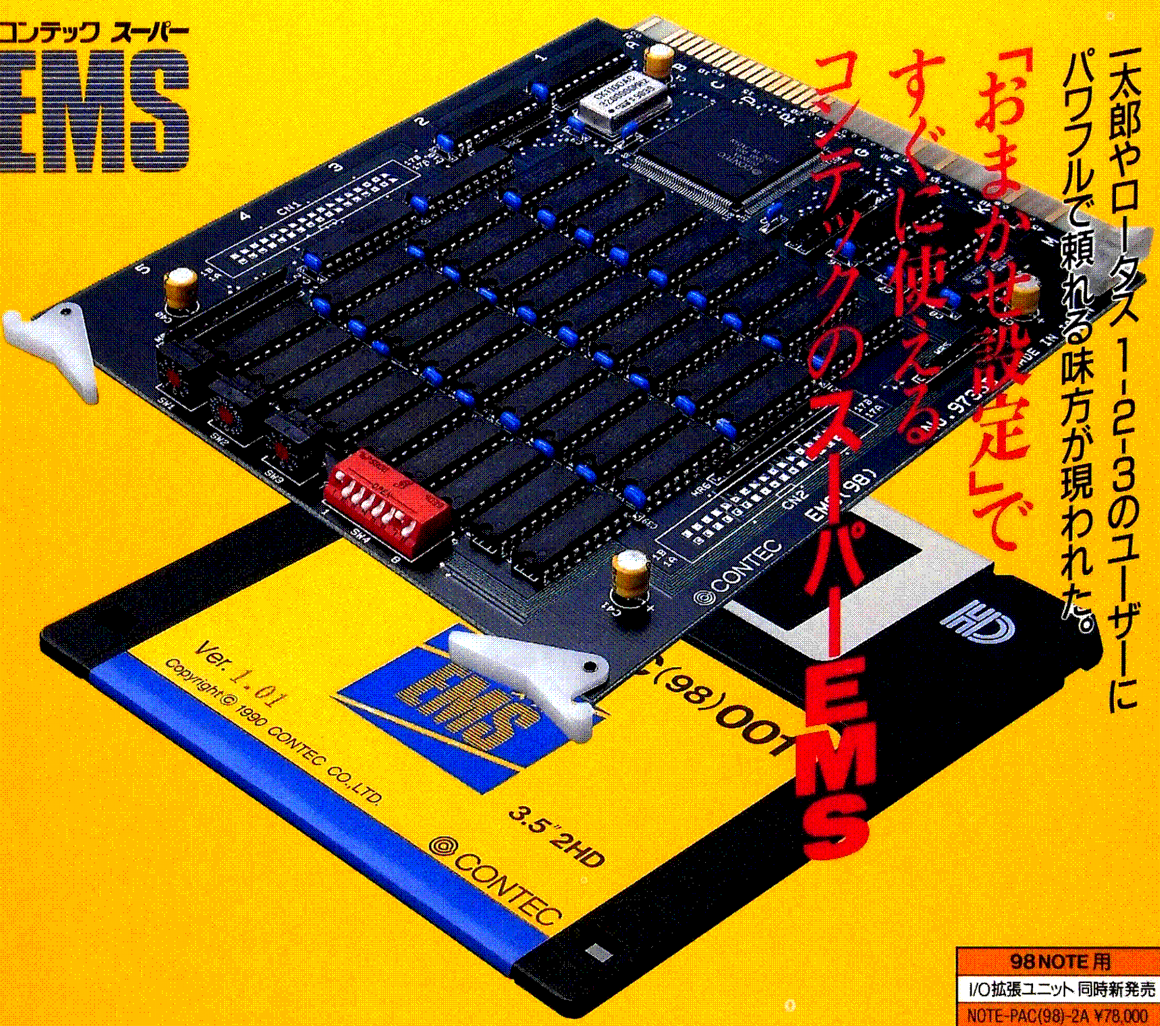
Get PSP (PSPを取得する)
(呼) AH=62H
(返) BX=カレントプロセスのセグメントアドレス

コンテック スーパー

EMS

一太郎やロータス1-2-3のユーザーに
パワフルで頼れる味方が現われた。

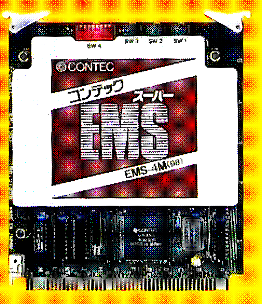
「おまかせ設定」で
すぐに使える
コンテックのスーパーEMS



「おまかせ設定プログラム」でらくらく導入

パワフルで高速なEMSを使いたい、しかし、難しい設定はどうも苦手、という方におすすめするのがコンテックの「スーパーEMS」。付属の「おまかせ設定プログラム」を使うと、すべての設定操作をプログラムが自動的に処理。難しい理屈はいっさい抜きにして、たちまちEMSの設定が完了します。どなたでも購入したその日から、EMS環境を実現できるのです。

用途に合わせて細かに設定したい方は、もうひとつの「設定プログラム」をお使いください。自動、簡単、詳細の3つのレベルにより、メモリ配置等すべてにわたり設定することが可能です。長年の実績と技術力を結集し、3つのスーパーをコンセプトに研究・開発した結果生みだされた、誰でも扱えるEMSボードが——「スーパーEMS」なのです。



一太郎は株式会社ジャストシステム、Lotusおよび1-2-3は米国ロータス・ディベロップメント社、MS-Windowsは米国マイクロソフト社の登録商標です。

Super Friendly ユーザーに優しい設定プログラム
Super High Quality FA、LAで培った技術が支える高い品質
Super High Speed 長年にわたる技術の蓄積が生みだした高速な実行スピード

メモリ不足の解消に!

- **EMS(LIM4.0)** 一太郎やLotus1-2-3などのソフトがスムーズに実行できます。
- **XMS(HMA)** MS-Windowsなどがスムーズに実行できます。
- **仮想EMS** プロテクトメモリを使用して、EMSと同じ機能を実現できます。
- **仮想UMB** プロテクトメモリを拡張ROM空間に割り当てることができます。
- **メモリ増設** メインメモリ、バンクメモリ、プロテクトメモリ、ハイレゾプロテクトメモリとして増設できます。

ファイルアクセスのスピードアップに!

- **RAMディスク** RAMを高速なディスクとして使用できます。
- **ディスクキャッシュ** フロッピーディスクやハードディスクを、より高速にアクセスできます。

いま時代は、EMS環境を求めています。

型 式	容 量	価 格
EMS-2M(98)	2MByte	¥37,000
EMS-4M(98)	4MByte	¥63,000
EMS-6M(98)	6MByte	¥89,000
EMS-8M(98)	8MByte	¥115,000

●5" 2HD, 3.5" 2HD両メディア添付

対応機種	
NEC	PC-9800シリーズ
エプソン	PC286,386シリーズ

■MS-Windows対応
■LANボードとの混在可

●スーパーEMSの技術的なお問合せは
フリーダイヤル
0120-00-9804
FAX 0472-96-1060

FA&OA
CONTEC
株式会社コンテック

コンテックFAマイコンセンター
〒555 大阪市西淀川区姫里3-9-31
TEL.(06)472-0265 FAX.(06)478-1031

NEC マイコンショップ
コンテック マイコンセンター

T4911777707985 〈資料請求番号P04〉 雑誌コード17777-07